NISTIR 4703

Technology Integration Workshop: Selected Papers

Henry Tom
Editor

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

NIST

NISTIR 4703

Technology Integration Workshop: Selected Papers

Henry Tom
Editor

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

October 1991

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

ABSTRACT

This report contains selected summaries of technical presentations and demonstrations given at the National Institute of Standards and Technology (NIST) Geographic Information Systems (GIS) Standards Laboratory's Technology Integration Workshop held at NIST, Gaithersburg, Maryland on August 23-24, 1990. The Workshop hosted over 50 representatives from a wide variety of governmental, industrial, and academic organizations and generated considerable interest and discussion among the attendees.

The Technology Integration Working Group, chaired by Mr. Dave Pendleton, NOAA, was formed within the GIS Lab as a cooperative technology transfer vehicle to share advances being made in applying expert systems, object-oriented database technologies, and GIS to practical problems in spatial data management and cartographic portrayal. The Workshop focused upon demonstrations of work-in-progress prototypes and technical discussions of progress in several on-going projects.

This Workshop as well as other NIST GIS Standards Laboratory activities are focused on performing cooperative research in order to integrate existing, emerging, and the anticipatory development of spatial data and information technology standards. The forum of government, industry, and academic organizations participating in the NIST GIS Standards Laboratory is dedicated to achieving this objective.

INTRODUCTION

Geographic Information Systems (GIS), expert systems, and object-oriented techniques are leading edge technologies. Substantial research in each technology and the integration of these technologies is occuring. Such integration efforts have been limited to just combinations of any two of these technologies.

The integration of all three technologies is being pioneered at the National Oceanic and Atmospheric Administration (NOAA) Nautical Charting Research and Development Laboratory (NCRDL), a component of the National Ocean Service's Office of Charting and Geodetic Services. During the past four years, the NCRDL has developed a series of prototypes for the autonomous generation of navigational charts from digital databases of chart features. The demonstrations and presentations were in large part based on the results of these prototype projects and their supporting hardware and software packages.

The prototypes were grounded in the artificial intelligence and object database technologies to effectively integrate the expert systems, GIS, and object-oriented programming/data management approaches. The work is a technological melange; each approach contributes to the overall solution with no individual approach capable of complete solutions by itself. The integration of leading-edge technologies for such applications is required because conventional methods have proven inadequate for the complexities of knowledge representation and reasoning about spatial entities.

The achievement of autonomous chart generation requires solving several classical digital cartographic problems, including: (1) feature generation (collapse and agglomeration) at scale change; (2) feature label placement without overlapping nearby feature symbology; (3) spatial conflict detection and resolution between nearby features at a given scale; (4) decluttering to emphasis features of special interest; and (5) graphic portrayal for a wide variety of complex features. Solving basic problems, common to the digital charting and GIS communities, has been the motivation behind the demonstrations and techniques discussed at the Workshop.

My thanks and appreciation to all the authors and attendees for their interest and participation.

Henry Tom
NIST GIS Standards Laboratory

Table of Contents

EXPERT SYSTEM CONCEPTS

by

Christopher E. Dabrowski
Elizabeth N. Fong

This paper provides an overview of the main concepts that underlie expert systems technology and discusses how this technology can be applied in practice. The paper is tutorial in nature and is intended for readers with little or no knowledge of expert systems. The paper presents the major classes, or kinds, of expert systems that can be developed and discusses criteria for selecting candidate tasks for implementation.

This paper provides an overview of the main concepts underlying expert systems technology and discusses how this technology can be applied in practice.

The overview presented in this paper is intended to be introductory in nature. The paper first discusses underlying concepts and reviews the computing methods employed by expert systems. An example of an expert system in operation is presented. Differences between expert systems and conventional software systems are summarized. Finally, the subject of interfaces between expert systems and other software systems is also covered.

The discussion of applications of expert systems technology begins by presenting the major classes, or kinds, of expert systems. This is followed by a discussion of criteria for selecting tasks for implementation.

Other aspects of expert system technology not covered in this paper include the expert system development process and software tools for developing expert systems. Readers interested in these topics can consult the sources listed at the end of this report.

1. INTRODUCTION

Recent years have seen a modest boom in the development of expert systems, both in government and industry. Initially, expert systems were advertized by some as holding forth the promise of revolutionizing the computer technology. Some hoped that expert systems could provide computers with a general capability for intelligent problem solving that could be applied to almost any problem. Such predictions proved to be too optimistic. As the technology matured, these views have gradually been corrected. More and more, expert systems are being realized for what they are: specialized software systems for automating expert problem solving for specific types of problems.

1.1 Background

Expert systems originated as part of the field of artificial intelligence. Artificial intelligence (AI) can be described as the study of theories and methods for automating "intelligent" behavior. AI has been a research discipline for over three decades, with other major subfields in robotics, machine vision, and natural language understanding.

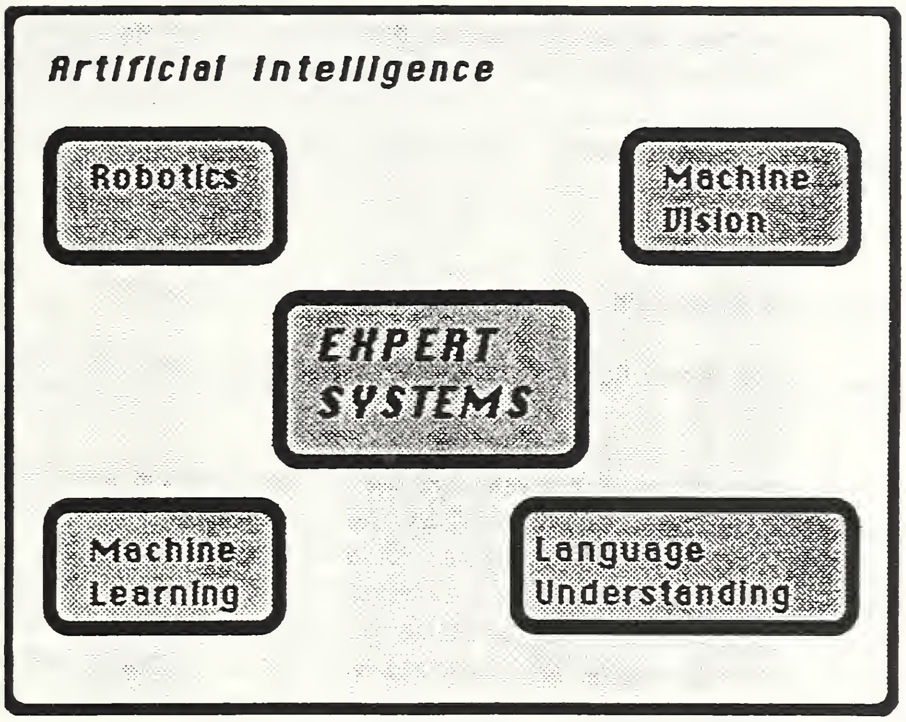


Figure 1. Expert Systems and Artificial Intelligence

The first research expert systems were developed in the late 1960s and early 1970s. MYCIN, an experimental system for diagnosis of blood diseases was one of the first successful demonstrations of the potential of expert systems. The early 1980s saw the first practical applications of expert systems in real-world environments. Expert systems have been developed to do medical diagnosis, determine causes of machine failure, to perform planning and scheduling activities, and to configure computer systems.

1.2 What Are Expert Systems

An expert system is a computer program for solving difficult problems that are normally handled by human experts. An expert system stores knowledge about how a particular type of problem is solved. When an example of the problem is presented, the expert system uses the stored knowledge to find a solution.

Expert system programs differ from conventional computer programs. Conventional programs are designed to solve problems in which all the relevant factors can be completely analyzed. For such problems, algorithms either exist or can be developed that perform a complete analysis of each aspect of the problem and arrive at a correct solution. Such algorithms can be encoded using a conventional programming language.

In contrast, expert systems are aimed at problems that cannot always be solved using an algorithm. These problems are often characterized by irregular structure, incomplete information, and considerable complexity. The solutions may be uncertain and represent "best guesses" that must be inferred on the basis of available evidence.

1.3 The Importance of Expertise

Expert systems are specifically designed to maintain and use human problem solving expertise.

Expertise can be considered knowledge about solving a special problem or a well-developed skill at performing a particular task. General expertise within a domain of knowledge can be partly acquired through formal training or education. However, in many domains of endeavor, extensive experience is needed to achieve a high level of performance in solving specific problems. Typically, this experience is possessed by only a few humans, who are referred to as domain experts. Much of the problem solving ability of domain experts is based on heuristics.

A heuristic can be regarded as a rule of thumb, fact, or even procedure that can be used to solve some problem but is not guaranteed to do so. It may fail. Heuristics may be regarded as simplifications of a complete formal description of some real-world system. For example, it is conceivable that a weather system can be described by a complex mathematical model. In principle, such a model could also be used to analyze atmospheric conditions and accurately predict the weather. However, in practice, complete models are often difficult to develop, due to lack of necessary information about the problem and its inherent complexity. Instead of using this model to predict rain, a simple heuristic can be substituted.

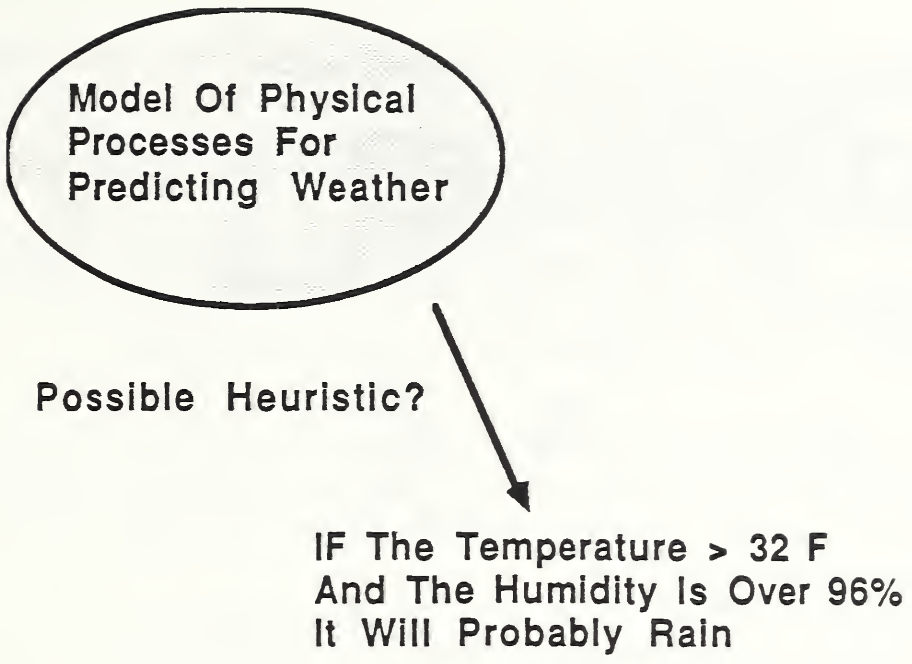

Model Of Physical Processes For Predicting Weather
Possible Heuristic?
IF The Temperature > 32 F
And The Humidity Is Over 96%
It Will Probably Rain

Figure 2. An Example of a Heuristic

The rain heuristic, while it will work most of the time, can also fail. However, the heuristic may be necessary if using a complex, formal model is not practical, or possible. In the real world, there are many important problems that cannot be solved using formally defined procedures. For these problems, using human expertise based on heuristic knowledge is necessary.

Although they can and do incorporate different kinds of programming methods, expert systems specialize in encoding and using heuristic knowledge. Expert systems can be thought of as software systems for 1) maintaining heuristic knowledge about certain kinds of problems, and 2) applying this knowledge to solve specific problems, usually using some form of automated inference. The expert system's problem solving ability is a function of the quality and quantity of knowledge it internally represents and uses. To internally represent knowledge and carry out inference, expert systems utilize specialized programming techniques developed through AI research.

2.0 THE ELEMENTS OF AN EXPERT SYSTEM

Expert systems store expert knowledge and apply it "on demand" to solve problems. Information about problems is provided to the expert system by human users, known as end users, via a computer terminal (the user of an expert system may also be another computer program or mechanical device as will be described below). The expert system uses inference procedures to apply its stored knowledge to facts describing problems. The systematic application of inference produces a solution that is displayed at the terminal.

The operation of an expert system is based on the interaction of three components. The knowledge base stores knowledge about how to solve problems. The inference procedures are applied by a software component of the expert system, called the inference engine. Communications between the expert system and end users are handled by an end user interface.

Figure 3 provides a graphical illustration that summarizes the architecture of the expert system.

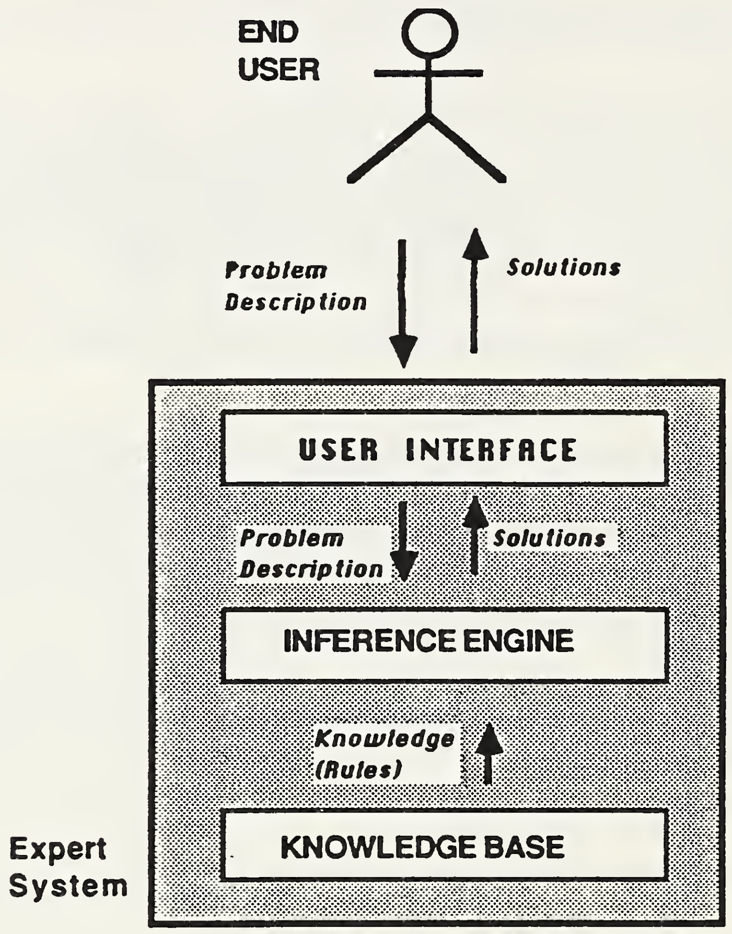


Figure 3. The Architecture of an Expert System

Each of the major architectural components of the expert system can be further elaborated.

2.1 The Knowledge Base

Knowledge is stored in the knowledge base using symbols and data structures to represent important concepts. The most common form of representing knowledge is the IF-THEN rule, or production rule, shown below.

```
Rule 1
IF      Days_Since_Rain in ?area >= 8
      & Humidity in ?area < 20%
THEN Conditions in ?area are DRY
```

Figure 4. A Typical Knowledge Base Rule

This rule expresses knowledge about conditions under which wildfires are likely. The symbol "?area" represents a variable that can take the value of a specific geographic area. The example will be elaborated below.

A knowledge base can contain many rules. Knowledge expressed in rule form is said to be stated declaratively. Declarative knowledge is knowledge that is stated explicitly. Declarative knowledge is intended to be accessible to persons, such as domain experts, who may need to see it. The ability to make its knowledge accessible and understandable is one of the most important services provided by an expert system.

2.2 The Inference Engine

The inference engine is a computer program that contains a set of inference procedures for applying knowledge stored in the knowledge base to a set of facts about a problem. These inference procedures are designed to produce new information leading to a solution.

An inference engine works by comparing rules against known facts to determine if new facts can be inferred. The conditions in the premise, or IF part, of a production rule are compared against existing known facts. If these conditions are satisfied, the facts in the conclusion, or THEN part, can be inferred. Figure 5 below illustrates how inference takes place.

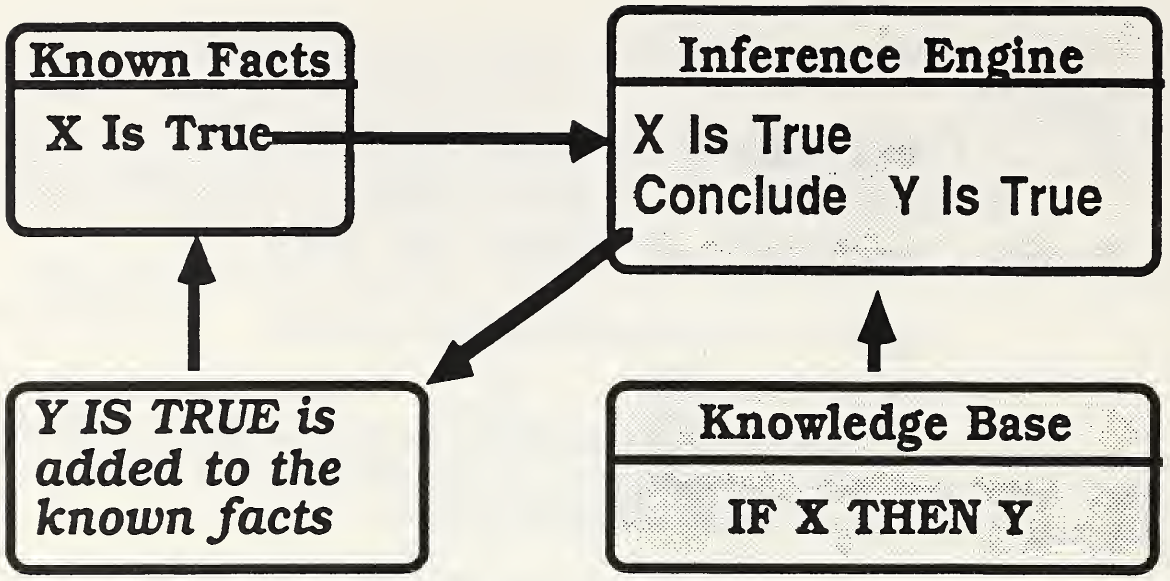


Figure 5. Rules and Inference

In an expert system, inference is said to take place through symbolic reasoning. An inference engine reasons symbolically by manipulating symbols that represent concepts. Symbolic reasoning is intended to emulate the way humans might manipulate concepts and ideas when reasoning. Many methods of symbolic reasoning are possible. The example presented above is a simple case. Symbolic reasoning is one way in which expert systems can be distinguished from conventional programs.

2.3 Inference Engines and Rule Chaining

During an individual problem solving session, or consultation, a large number of rules may be examined. The inference engine uses the fact or facts concluded by one rule to satisfy conditions of other rules. In this way, many different rules can be linked or chained. Figure 6 provides a graphic illustration of how rule chaining works.

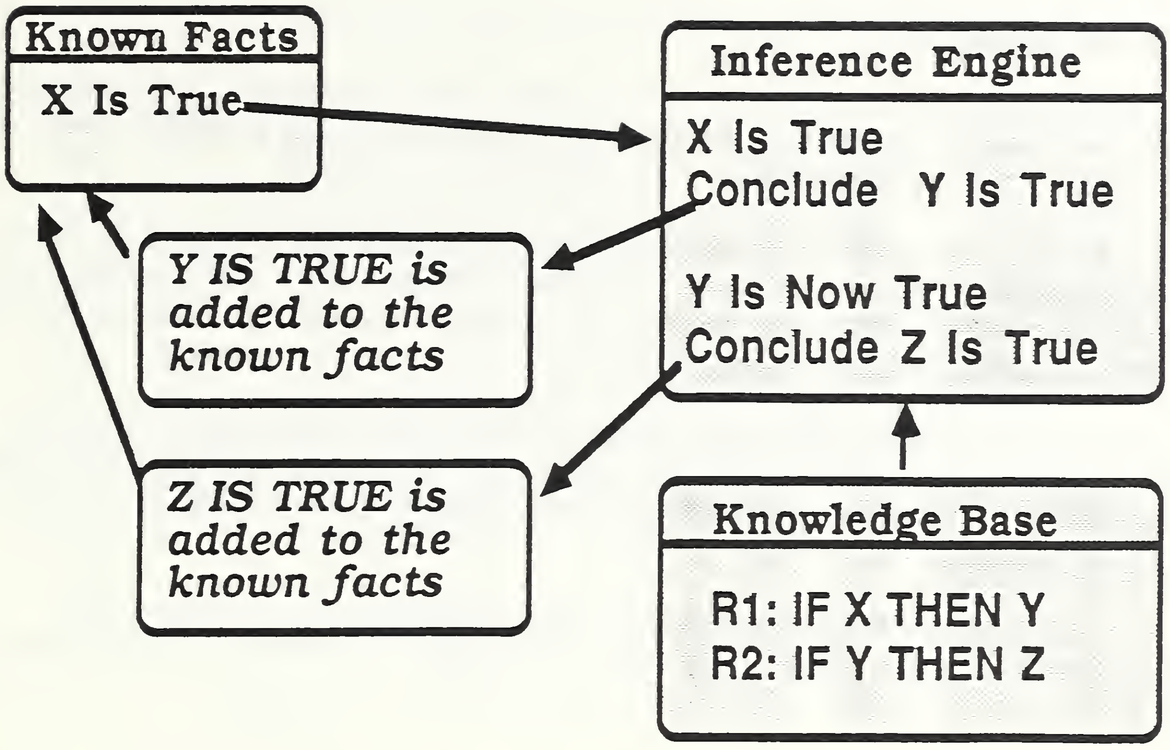


Figure 6. Inference Engines "Chain" Rules

Inference engines chain rules using two strategies. In forward chaining, known facts are compared to rule conditions to determine what rules can be satisfied. Figure 6 shows a simple example of forward chaining. The forward chaining process repeats until a problem solution is reached, or until no new facts can be concluded. Forward chaining is said to proceed in a forward direction because a fact is first inferred by one rule and then compared against conditions of other rules in the knowledge base.

Backward chaining works the opposite way. In backward chaining, the inference engine first selects a fact that it intends to infer. Initially, facts representing solutions to a problem are selected. Next, the inference engine finds rules that infer the desired fact and attempts to satisfy their conditions. The conditions of these rules may require facts that are unknown to the inference engine. These unknown facts may be concluded by different rules that must then be examined and so forth.

In backward chaining, rules are chained in backward direction, proceeding from the unsatisfied conditions of one rule to rules that conclude the necessary facts. Like forward chaining, backward chaining can involve many rules. Inference engines employ control strategies to guide the order by which rules are examined and inferences made. For more detailed explanations of backward and forward chaining, see [RICH91] or [WINS84].

2.4 External Interfaces

Expert systems communicate with human users via an end user interface. The end user interface is used to obtain information about the problem from the end user and to display solutions.

To do this, an interface must display questions at a terminal and provide prompts or menus to allow end users to answer. The solutions may be displayed using text. An expert system for geographic information systems (GIS) applications might use computerized maps, if appropriate.

The user interface must also provide explanations of the expert system's actions. During a consultation, the user may wish to know why questions are being asked or why certain facts were concluded. At the end of a consultation, the user may request explanations of how a solution was reached.

For many practical applications, an expert system must interface, and exchange data with other software systems. This topic will be discussed further in section 4.

3. AN EXAMPLE OF AN EXPERT SYSTEM

This section provides an idealized example of how an expert system can be employed in practice. The example is based on a typical GIS application: use of geographic information to determine the likelihood of forest fires.

3.1 A Sample Knowledge Base

Let us assume a simplified knowledge base that has two rules, shown in figure 7 below. Rule 1 is the same rule shown in figure 4. Rule 2 concludes that a forest fire, called a "wild fire" is likely if 1) lightning strikes have been seen in the area and 2) conditions in the area are dry.

```
Rule 1
IF      Days_Since_Rain in ?area >= 8
     & Humidity in ?area < 20%
THEN Conditions in ?area are DRY

Rule 2
IF      Lightning_Seen in ?area
     & Conditions in ?area are DRY
THEN Wildfire in ?area LIKELY
```

Figure 7. A Small Knowledge Base of Rules

3.2 A Sample Problem Solving Session

The expert system is activated by an end user, perhaps a forest ranger. The ranger wishes to know where within a park a "wildfire" likely to occur. For the purposes of this example, assume the park is divided into areas. Also assume that the ranger is concerned with only 2 particular areas: AREA_1 and AREA_2. The ranger has access to information about weather conditions in these areas, including the number of days since the last rain, the humidity, and the occurrence of lightning strikes. The ranger provides these facts to the expert system via the end user interface.

In this example, the inference engine attempts to provide an answer to the ranger's problem using forward chaining. The process begins by comparing the known facts provided by the ranger against the conditions of rules.

1. The known facts are compared to the conditions of Rule 1. The facts Days_Since_Rain = 14 and Humidity = 5% for AREA_1 satisfy the conditions of Rule 1. The fact "Conditions in AREA_1 are DRY" can be inferred.

2. For AREA_2, The facts Days_Since_Rain = 6 and Humidity for AREA_2 = 60% do not satisfy the conditions of Rule 1. Therefore, the inference engine cannot conclude that conditions are dry in AREA_2.

3. The known facts are compared against the conditions of Rule 2. Again, Rule 2 can be satisfied for AREA_1. Conditions in AREA_1 are

DRY was inferred by Rule 1. Lightning_Strikes_Seen in AREA_1 is a known fact. Therefore, Wildfire in AREA_1 can be concluded.

4. The ranger receives an answer that a wildfire is likely in AREA_1.

The process is summarized in figure 8 below.

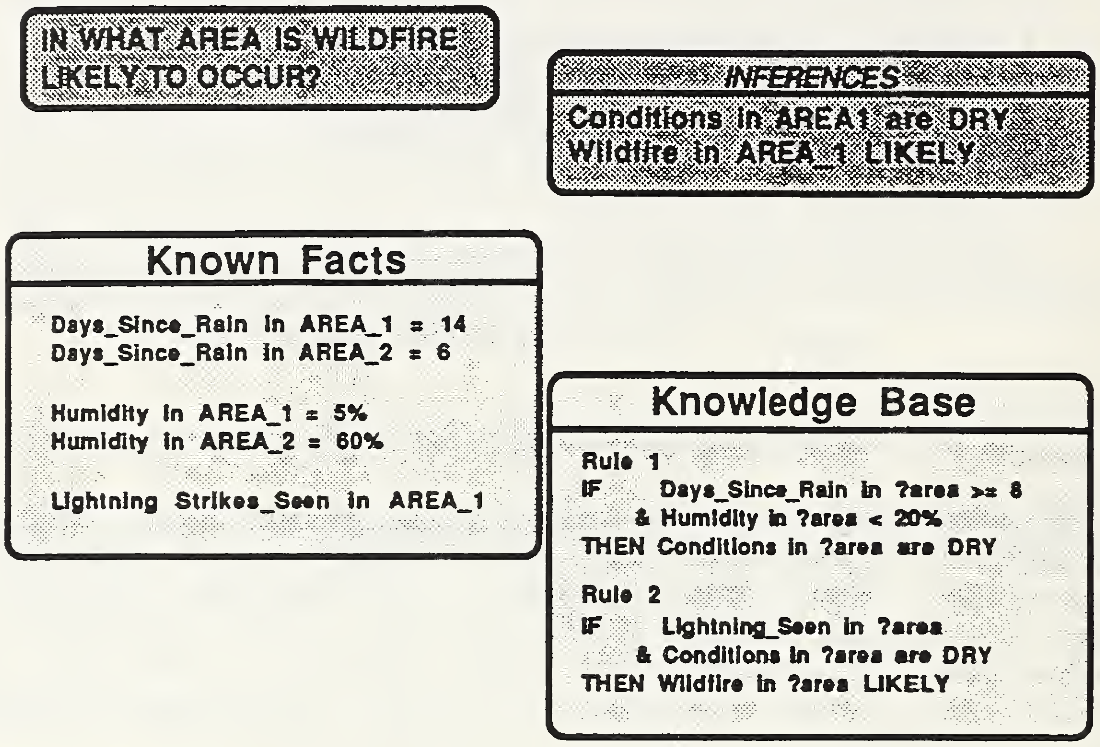

IN WHAT AREA IS WILDFIRE LIKELY TO OCCUR?
INFERENCES
Conditions in AREA1 are DRY
Wildfire in AREA_1 LIKELY
Known Facts
Days_Since_Rain in AREA_1 = 14
Days_Since_Rain in AREA_2 = 6
Humidity in AREA_1 = 5%
Humidity in AREA_2 = 60%
Lightning Strikes_Seen in AREA_1
Knowledge Base
Rule 1
IF Days_Since_Rain in ?area >= 8
& Humidity in ?area < 20%
THEN Conditions in ?area are DRY
Rule 2
IF Lightning_Seen in ?area
& Conditions in ?area are DRY
THEN Wildfire in ?area LIKELY

Figure 8. A Simple Example: Predicting Fires

In backward chaining, the inference engine would first find a rule that concludes a solution to the problem; e.g., there is a likelihood of wildfire in some area. Therefore, Rule 2 would be tried first. The conditions of Rule 2 are initially unsatisfied because the fact stating that the area is dry is unknown. The inference engine would then have to test Rule 1 to conclude which area was dry.

This simple example is meant to give the reader some idea of how an expert system works. Actual expert systems may have hundreds or even thousands of rules.

3.3 Real-World Expert Systems and Problem Complexity

The difficulty of many real-world problems tackled by expert systems can often be described in terms of: (1) the large number of facts that can potentially be examined, (2) the many different solutions that can be developed, or both. Examining the consequences of each fact or combination of facts and investigating every possible solution could result in following many alternative lines of reasoning. The systematic examination of different lines of reasoning in an effort to find a solution can be viewed as a search process. For very large problems, exhaustively searching every possibility might be impractical. Solving such problems thus depends on reducing their size.

In many cases, the expert limits the size of the problem by focusing on a small subset of facts and a few relevant lines of reasoning. This allows the expert's knowledge to be captured in a succinct set of rules. The resulting "expert systems" are small and the knowledge they contain can be expressed in decision tables.[1]

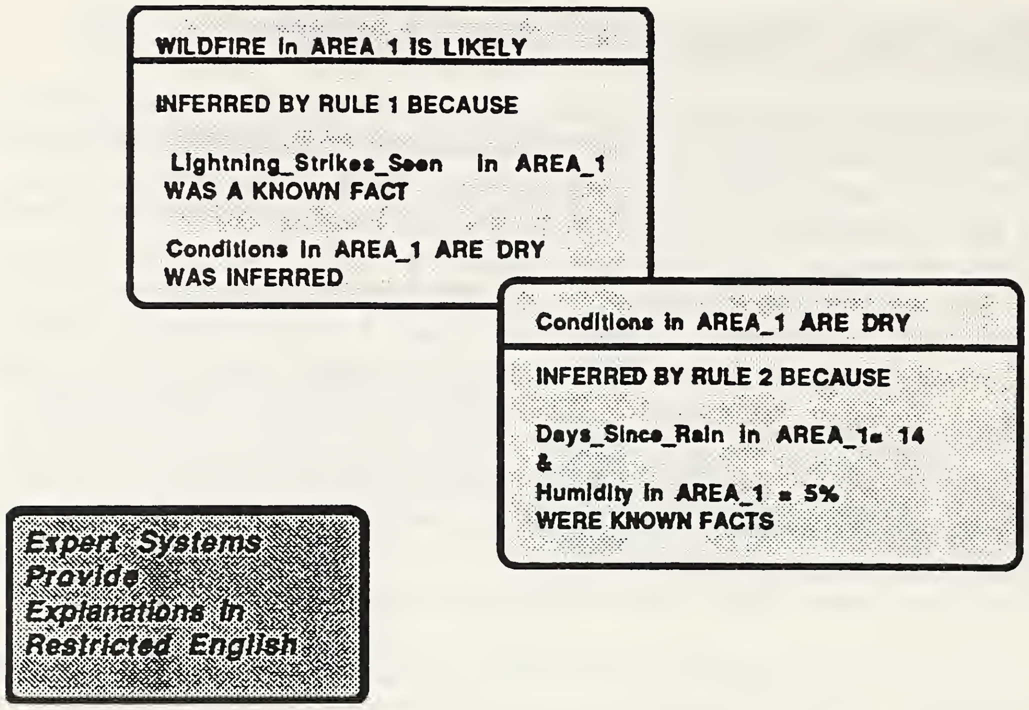
For larger, more complex problems, finding a solution requires examining more information and/or considering a much larger number of solutions. To reduce the size of a problem, an expert system may have to select only a few lines of reasoning to pursue from a much larger number that may be available. Heuristics provided by the expert must be used to select those lines of reasoning with the best chance of producing a solution.

The use of heuristics to limit search is known as heuristic search. One way heuristics can reduce search is to shift the focus of the problem-solving effort on the most important evidence or the most likely solutions. In the example above, the domain expert may provide the knowledge that the machine is more likely to overheat than it is for the cooling system to fail. Hence, the expert system could reduce the amount of work it has to do by trying Rule 2 first. Other lines of reasoning would be tried if it was found that the machine had not overheated. In a larger expert system, heuristic search could be an important factor. For more information about search in AI, see [FOX90], [RICH91], or [WINS84].

3.4 How Expert Systems Explain Their Actions

It is helpful for an expert system that solves a complex problem to be able to explain its actions. For instance, in the example discussed in section 3.2, the end user may be interested in an explanation of the line of reasoning used to determine that "wildfire" is likely in AREA_1. Figure 9 shows how an expert system might display its line of reasoning.

[1]Such systems are often distinguished from expert systems because of their simplicity and small size. Some specialists prefer to call these systems knowledge systems.


WILDFIRE In AREA 1 IS LIKELY

INFERRED BY RULE 1 BECAUSE

Lightning_Strikes_Seen In AREA_1
WAS A KNOWN FACT

Conditions In AREA_1 ARE DRY
WAS INFERRED

Conditions In AREA_1 ARE DRY

INFERRED BY RULE 2 BECAUSE

Days_Since_Rain In AREA_1= 14
&
Humidity In AREA_1 = 5%
WERE KNOWN FACTS

Figure 9. Explain Why Wildfire Is Likely in Area 1.

An expert system keeps track of what facts have been inferred and which rules were used to make the inferences. As the figure shows, explanation of how facts are concluded is achieved by "playing back" the sequence of inferences that were made.

3.5 Differences Between Expert Systems and Conventional Computer Programs

Expert system programs differ from conventional software in four important ways. First, knowledge is separated from program control; e.g., the knowledge base and inference engine are separate. Second, knowledge is represented declaratively. Third, expert systems perform computation through symbolic reasoning. And finally, expert systems can explain their actions. Each of these distinguishing capabilities can be elaborated.

o **Separation of Knowledge From Program Control**

In conventional programs, written in languages such as FORTRAN or C, knowledge about a problem domain is contained in programming language statements. As such, knowledge about how a problem is solved is combined directly with specifications for control of program execution. In an expert system, knowledge is represented in data structures, such as production rules, which are stored in a knowledge base. The knowledge base is separate from the inference engine which controls program execution.

- **Declarative Representation of Knowledge**

  Knowledge represented using symbols and data structures, such as rules, is explicit in the sense that it states what knowledge exists, not how the knowledge is applied. By representing knowledge declaratively, the knowledge of an expert system can be more readily understood and accessed by individuals who are not specialists in programming languages.

- **Symbolic Reasoning and Inference**

  Symbolic reasoning refers to the manipulation of symbols and data structures by the inference engine. Symbolic reasoning is intended to emulate the way humans might manipulate concepts when solving a problem. In the simplest case, this may involve testing a rule to determine if its condition part is satisfied and then adding the conclusion of the rule to a list of known facts. Expert systems can also be characterized by use of defined inference strategies.

- **Explanation of Actions**

  Declarative knowledge and symbolic reasoning support the ability of the expert system to explain its actions. An expert system explains its actions by showing the chain of reasoning created by the rules used to solve a problem.

The ability to represent and reason about uncertainty is a feature also supported by some expert systems.

4.0 EXPERT SYSTEMS AND OTHER SOFTWARE SYSTEMS

As more and more corporate information resources become computerized and as the demand for expert systems increases, the necessity and importance of expert system interfaces has also increased. Many problem solving tasks performed by expert systems require communication, and exchange of data, with other software systems. This section briefly discusses this important subject.

4.1 Expert System Interfaces to Other Software

Expert systems may require a variety of external interfaces. A few are summarized below.

- **Interfaces to Database Management Systems**

  Expert systems often require access to data residing in a DBMS or GIS. For instance, in the example above, information about conditions and local weather can be conveniently stored in a database. Instead of obtaining this information from a ranger, the expert system might initiate a call to a DBMS.

- o **Interfaces to Procedural Computer Programs**

  While expert systems carry out inference procedures well, they are not as good at performing other computations. For instance, inference procedures are inefficient for extensive numerical calculations or iterative processing. If such computations are required, the expert system should initiate calls to external software modules written in languages such as C or Fortran. An expert system that uses GIS analysis functions must operate in this manner.

- o **Graphics Systems and Packages**

  Interfaces to graphics software packages may be necessary for many applications of expert systems. Graphics packages are integral aspects not only of GIS, but also of engineering software such as computer aided design (CAD) systems.

  Figure 10 depicts various types of interfaces an expert system may have.

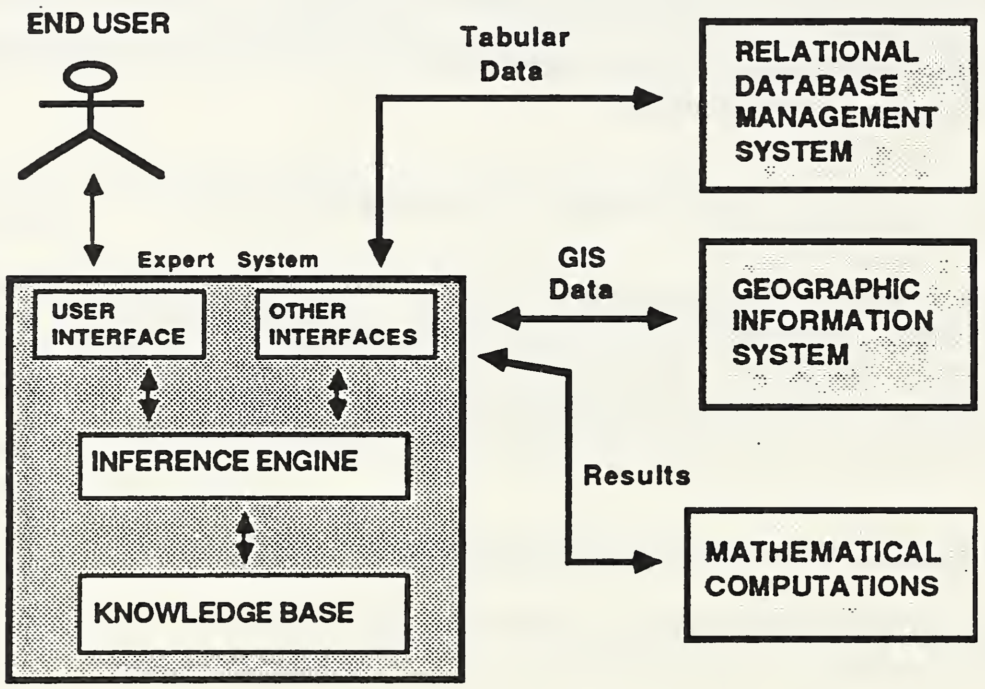


Figure 10. Expert System Interfaces

4.2 Embedded Expert Systems

It is possible that conventional software systems, including GIS applications, may initiate calls to expert systems to perform special purpose problem solving. Increasingly, expert systems are being developed and configured as components in larger software applications. Such expert systems are referred to as embedded expert systems. The primary user of an embedded expert system is another software application. Figure 11 below illustrates this concept.

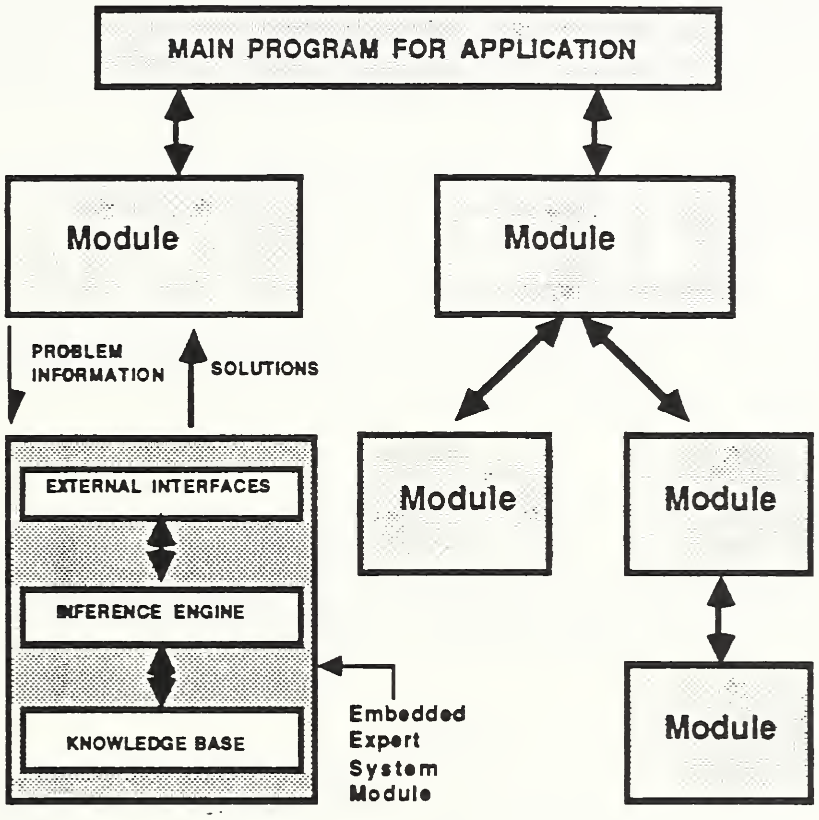


Figure 11. Embedded Expert Systems

5. APPLICATIONS OF EXPERT SYSTEMS

Expert systems are specialized software systems that emulate human reasoning activity. But, exactly what kinds of applications can expert systems best be used for? The next two sections (secs. 5 and 6) will help answer this question.

Expert systems employ two broad categories of problem solving strategies. They are solution selection and solution construction. These strategies provide a basis for describing classes of expert system applications.

5.1 Expert Systems That Select Solutions

The most basic strategy employed by expert systems is to select solutions from among a list of possible outcomes. Selection systems work by essentially matching problem descriptions; e.g., matching facts describing the circumstances of a problem to individual solutions.

More complex problems require use of abstract problem characterizations as intermediate steps in finding a solution. The expert system examines supporting evidence in an attempt to characterize, or classify, a problem according to abstracted descriptions of problem types. The characterizations are then used to infer solutions. The abstract characterizations and rules of inference are based on heuristics obtained from domain experts. This method, known as heuristic classification, summarizes the way many expert systems solve problems [CLAN85].

Several kinds of expert system applications that select solutions using heuristic classification may be identified. A few are listed below.

- **Recognition of System Malfunctions**

  Determining causes of malfunctions is a generic task commonly performed by expert systems. Examples include medical diagnosis and troubleshooting equipment failure.

- **Evaluation of Applications for Benefits**

  These systems determine eligibility for credit applications, insurance claims, and pensions. There are several examples of such systems implemented on microcomputers. See [HARM88].

- **Selection of Items From a Catalog**

  In many organizations, expert systems are used to select goods or services from a larger list of possibilities to meet specific user needs [POTT90], [FONG88], [RADA90].

- **Monitoring and Control**

  System monitoring, closely related to diagnosis, compares actual system behavior to a model of preferred behavior, operating on a continuous basis [TSUD90].

Other kinds of expert systems that select solutions may be identified. See also [HAYE83], [CLAN85]. Selection systems vary in complexity, from simple product selection systems to complex diagnostic systems.

5.2 Expert Systems That Construct Solutions

A second method employed by expert systems is to construct solutions from individual components or pieces. This method is used to perform configuration, design, planning, or scheduling tasks.

Expert systems that construct solutions must first select individual components (or have them provided in the problem description). Components may have complex relationships, many of which reoccur in generic patterns. Similarly, constraints may be placed on relationships. The knowledge base may contain generalized rules that specify allowable arrangements of components or that express constraints.

The problem solving strategy of some moderate to large construction type systems requires consideration of many alternative component combinations. For larger configuration problems, the number of potential combinations can exceed the computational resources of the largest computer systems. Many construction expert systems find solutions by employing heuristic search strategies to reduce the number of combinations that need to be examined (See sec. 3.3). Examples of construction tasks are listed below.

- **Design**

  Expert systems have been developed to perform a number of design tasks including designing maps [ROBI85], [NICK86], configuring hardware components of computer systems to satisfy customer requirements [MCDE82], [MCDE84], physical database design [DABR88], and other areas [LIEB90].

- **Planning and Scheduling**

  The goal of expert systems that plan and schedule is to find an acceptable arrangement of events so as to minimize use of time and resources. A typical example is finding optimal travel routes for delivery trucks [ROTH90].

Most, though not all, construction-oriented tasks require extensive processing. Small scale configuration tasks can be largely procedural and often do not require extensive computing to perform.

6.0 SELECTING APPLICATIONS OF EXPERT SYSTEMS

Not all tasks can be implemented as expert systems. Some tasks rely on knowledge that cannot be easily captured in expert systems, while other tasks are better implemented using alternative computing methods.

This section discusses some criteria for selecting applications of expert system technology and examines where expert systems can be advantageously deployed. Characteristics of tasks that are appropriate for implementation as expert systems are presented. Tasks that are not appropriate for development as expert systems are also discussed.

6.1 Deploying Expert Systems to Improve Productivity

Determining how and where to use expert systems is based on analyzing the information processing needs of an organization. Identification of places to apply expert systems technology is based on finding critical points within an organization where automation of expertise can lead to improvements in operational efficiency.

o **Alleviating "Knowledge Bottlenecks"**

"Knowledge bottlenecks" occur when existing expertise cannot be brought to bear on regularly occurring problems that require expertise to solve. That is, "knowledge bottlenecks" happen when the number of experts is too small for the number of problems that need to be solved. Or, experts may be geographically distant from the site of the problem. The example in section 3 illustrates such a "bottleneck."

o **Providing a Means for Consistent Decision Making**

By deploying expert systems throughout an organization, expertise about narrowly focused problems can be disseminated. This ensures consistent implementation of policy and procedures.

o **Automating Repetitive Tasks That Are Difficult for Humans**

In many operational situations, certain tasks are performed by continuous repetition over long periods. One example is monitoring computer systems for illegal access attempts [TSUD90]. Such tasks require a certain level of expertise, but are performed poorly by people because they require constant attention. In these situations, expert systems have proved useful.

o **Freeing Experts for More Important Tasks**

An expert system can perform many relatively mundane tasks normally handled by an expert. This allows the expert to devote time to other work that may also be important to the organization.

6.2 Characteristics of Appropriate Problems

In determining whether or not expert system technology should be used, it is important to examine the characteristics of the problem to be solved. In practice, certain problems have proven more amenable for automation using expert systems methods than others. These problems generally possess certain characteristics.

o **Finding Solutions to Problems Requires Expertise**

The level of difficulty of the task must be high enough to pose some barrier to individuals who wish to become proficient in solving the problem. Finding solutions to problems depends on the use of expertise accumulated through training and/or practice.

o **Problem Solving in the Domain Is Uncertain**

Information about problem solving methods may be uncertain, imprecise, and incomplete. Uncertainty is a common characteristic of expert system domains, and expert systems provide methods to express different levels of confidence. A typical problem may have more than one solution, each of which has a degree of associated uncertainty.

o **The Problem Cannot Be Solved Using Established Computing Methods**

Analysis of the problem may show that it can be completely solved by mathematical techniques, operation research methods, or other established computing methods. In this case, expert system technology should not be used. Problem solving activity that is based on well-defined procedures, or involves only simple decision making, can often be more easily automated using conventional software methods.

6.3 Characteristics of Knowledge Used to Solve the Problem

Closely related to the characteristics of the problem to be solved, are the characteristics of the knowledge used to perform the problem solving task. These characteristics may be summarized as follows.

o **The Knowledge Necessary to Solve the Problem Depends on Heuristics**

Not all the knowledge that goes into an expert system is heuristic. But often, a large part of it is. An exception to this maybe applications such as claims benefits processing where most of the knowledge is legal or procedural and is not seen as being heuristic in nature.

- **The Knowledge Can Be Stated or Represented Declaratively**

  It must be possible to clearly state important concepts, rules, and procedures used to solve problems in the domain of interest. It must also be possible to express this information symbolically in data structures.

- **The Problem Solving Process Is Based on Reasoning and Deduction**

  As discussed earlier, expert systems are specifically designed to represent and reason with heuristic knowledge. In most cases, heuristics should be representable in rule form.

6.4 The Scope and Size of the Problem

In practice, the size of completed expert systems is often very large, consisting of hundreds, or thousands, of rules. If the task is too broad, the development effort may take an inordinate amount of time, or even be impossible. Some guidelines on the scope and size of the problem are listed below.

- **The Task Must Be Narrowly Focused**

  The problem to be solved must be restricted and specific. An expert system should be dedicated to a limited domain. For example, an expert system to diagnose factory machine failures may be dedicated to problems for a very specific kind of machine. Or, it may diagnose malfunctions of only very specific parts.

- **The Task Should Be Decomposable**

  A universal system development strategy is to decompose a system into components and develop each component separately. Tasks that lend themselves to the "divide and conquer" strategy can be implemented as expert systems in pieces, or phases. This greatly eases system development.

In general, selecting problems having sufficiently narrow scope is critical to successful development of expert systems.

6.5 Areas to Avoid

Several categories of problem solving activity have been identified as being inappropriate for expert system methods. These are listed below.

o **Tasks Based on Common Sense or Real-World Knowledge**

The amount of knowledge about the real world is so vast as to be virtually impossible to implement in a computer application. Representation of common sense knowledge is a topic of AI research.

o **Tasks Requiring Perceptual Knowledge**

Problems that require actually seeing or touching information about the problem are beyond the scope of expert systems.

o **Creativity or Inventiveness**

As stated above, expert systems are under a closed-world assumption and cannot be expected to engage in creative activity in the same sense as humans do.

Each of these categories involves processes that are too poorly understood to be implemented using current AI programming methods.

6.6 The Source of Expertise

To develop an expert system, an established source of expertise must exist. Without an existing base of expertise to solve the problem, expert system technology cannot be applied. The necessary expertise is possessed by a recognized human expert or is found in a written source, such as a manual. Sometimes expertise is obtained from both sources.

o **The Core of the Problem Solving Knowledge Must Be Stable**

While specific aspects of the problem solving knowledge might change over time, the underlying concept and basic problem solving method must remain stable. It is also important to emphasize that expert systems apply only existing expertise to solve problems. They do not create or invent expertise.

o **There Must Be Substantial Agreement on Solutions Between Experts**

If several experts exist and disagree on the solutions for the most important or common problems encountered, this may indicate that the problem solving knowledge is unstable.

o **The Knowledge Provided by the Source of Expertise Must Be Clear**

The problem solving method and the underlying knowledge must be understandable to developers.

o **The Domain Expert Must Be Able to Allocate Time**

Too often the amount of time required to develop an expert system is underestimated. In most cases, it is necessary to have a domain expert assigned to a project for the length of its duration.

The availability of the source of expertise depends greatly upon management decisions about allocation of resources in an organization.

6.7 Summary and Further Sources on Selecting Expert Systems Applications

The discussion contained in this section can be summarized as a list of questions to consider when assessing a candidate task for development as an expert system.

o Can the task be clearly defined? Is the task narrow in focus?

o Does the task fall into one of the expert system categories described in section 5?

o Is there a benefit to the organization in automating the task? How will it help improve productivity?

o Does a domain expert exist who can perform the task? Can the domain expert clearly state the knowledge necessary to solve the problem? (If not, an expert system may not be possible.)

o Can the task be expressed in a straightforward algorithm that can be encoded using established computing methods? (If so, an expert system may not be necessary.)

o Does the task depend on heuristic knowledge? Does it require reasoning and inference?

o Does the task require skills that are difficult to automate? (If so, an expert system may not be possible.)

o Is the task small enough to automate? Is it decomposable?

Further references that deal with the subject of selecting expert system applications include [BECK90], [CUPE88], [HARM88], [LAUF90], [MOCK90], [MURD90], and [PRER89].

7.0 LIMITATIONS OF EXPERT SYSTEM TECHNOLOGY

The examples presented in this paper and previous discussions have indicated that expert systems have limitations. Some of these limitations are summarized below.

- **Expert System Technology Is Not Appropriate for All Problems**

  Suitable problems for development as expert systems were discussed at length in section 6 and will not be restated here.

- **Expert Systems Are Highly Specialized and "Brittle Around the Edges"**

  The problem solving power of an expert system is limited to those types of problems the knowledge base has knowledge about. If a problem is presented to the expert system that requires knowledge not contained in the knowledge base, even if the amount of this knowledge is minimal, the expert system will fail.

- **Maintenance of Large Expert Systems Is Poorly Understood**

  In practice, maintenance of large expert systems may prove costly and time consuming. Since few really large expert systems have been developed, this topic is poorly understood at present. Large expert systems will be an important area of research in the future.

8.0 CONCLUSIONS

This paper has introduced the basic concepts of what expert systems are and how they work. Applications of expert system technology and criteria for selecting appropriate tasks for development as expert systems were also discussed.

This brief introduction has emphasized both the benefits provided by expert system technology as well as some of the weaknesses and limitations. Despite limitations, expert system technology is both expanding rapidly and stabilizing. The technology is stabilizing in that much has been learned about how the technology can be used and what the limitations are. In many organizations, development of expert systems has become as acceptable as development of conventional software.

Expert system technology is expanding because it is being applied in many new domains of endeavor. Among the most important are Geographic Information Systems.

BIBLIOGRAPHY

[BECK90] Beckman, T. J. "Methods for selecting Promising Expert System Applications," The Fifth Annual AI Systems in Government Conference, May, 1990.

[CLAN85] Clancy, William J. "Heuristic Classification," in Knowledge-Based Problem Solving, Janusz Kowalik, (ed.), Prentice-Hall, 1986.

[CUPE88] Cupello, J. M., and Mishelevich, D. J., "Managing Prototype Knowledge/Expert System Projects," Communications of the ACM, Volume 31, Number 5, May, 1988.

[DABR88] Dabrowski, C. E., and Jefferson, D. K., A Knowledge-Based System for Physical Database Design, NBS Special Publication 500-151, National Institute of Standards and Technology, Gaithersburg, MD, February, 1988.

[FONG88] Fong, Elizabeth N., and Dabrowski, Christopher E., "An Expert System To Select Data Sources From Chemical Information Databases," ASME Computers in Engineering Conference, San Francisco, CA, August, 1988.

[FOX90] Fox, Mark S. "AI and Expert System Myths, Legends, and Facts," IEEE Expert, Volume 5, Number 1, pp. 8-20.

[HARM88] Harmon, Paul, et al. Expert Systems: Tools & Applications, John Wiley & Sons, 1988.

[HAYE83] Hayes-Roth, Frederick, and Waterman, Donald, and Lenat, Douglas (eds.). Building Expert Systems. Addison-Wesley, 1983.

[LAUF90] Laufmann, S. C., DeVaney, D. M., and Whiting, M. A., "A Methodology for Evaluating Potential KBS Applications," IEEE Expert, Volume 5, Number 6, December, 1990.

[LIEB90] Liebowitz, Jay. "Expert Configuration Systems: A Survey and Lessons Learned," Expert Systems With Applications, Volume 1, Number 2, pp. 183-187.

[MCDE82] McDermott, J. "R1: A Rule-based Configurer of Computer Systems," Artificial Intelligence, Volume 19, Number 1, 1982.

[MCDE84] McDermott, John and Bachant, Judith, "R1 Revisited: Four Years in the Trenches," AI Magazine, 5, (3), pp. 21-32, Fall 1984.

[MOCK90] Mockler, R. J. "Using knowledge-based systems for estimating risks inherent in a proposed KBS project," Expert Systems, February, 1990, Volume 7, Number 1.

[MURD90] Murdoch, H. "Choosing a problem – when is Artificial Intelligence appropriate for the retail industry?," Expert Systems, February, 1990, Volume 7, Number 1.

[NICK86] Nickerson, B. G., and Freeman, H., "Development of a Rule-Based System for Automatic Map Generalization," Proceedings of the International Symposium on Spatial Data Handling, pp. 537-556.

[POTT90] Potter, W. D., et al. "SLING: A Knowledge-Based Product Selector," Expert Systems With Applications, Volume 1, Number 2, pp. 161-169.

[PRER89] Prerau, D. S. "Choosing an Expert System Domain," in Topics in Expert System Design: Methodologies and Tools, Guida, G. and Tasso, C., North-Holland-Amsterdam, 1989.

[RADA90] Rada, R., "An Expert System for Journal Selection," IEEE Expert, Volume 5, Number 2, April, 1990.

[RICH91] Rich, E. and Knight, K. Artificial Intelligence, (second edition), McGraw-Hill, 1991.

[ROTH90] Roth, Janet. "NATIONAL DISPATCHER ROUTER: A Multi-Paradigm Based Scheduling Advisor," Second Annual Conference on Innovative Applications of Artificial Intelligence, Washington, D. C., May, 1990.

[ROBI85] Robinson G., and Jackson, M. "Expert Systems in Map Design," Proceedings of AutoCarto-7, pp. 430-439.

[SHOR76] Shortliffe, E. H., MYCIN: Computer-based Medical Consultations, Elsevier, New York, 1976.

[TSUD90] Tsudik, Gene and Summers, Rita. "AudES - an Expert System for Security Auditing," Second Annual Conference on Innovative Applications of Artificial Intelligence, Washington, D. C., May, 1990.

[WINS84] Winston, Patrick H. Artificial Intelligence. Addison-Wesley, 1984.

Geographic Information Systems and Expert Systems

Vincent B. Robinson, Director
Institute for Land Information Management
University of Toronto, Erindale Campus

Introduction

Geographic Information Systems (GIS) and Expert Systems (ES) are two dynamic technological domains. There has been much written on the potential to be realized through their integration (Robinson and Frank 1987; Robinson et al 1988; Fisher et al 1988). GIS and its application are fields fraught with complexity while ES provides tools and techniques for searching through complex problems to arrive at solutions.

Geographic Information Systems, Artificial Intelligence Expert Systems and Knowledgebased Systems

A Geographic Information System (GIS) is a specialized kind of information system designed to gather, process, and provide geographic information that may be relevant for research, management decisions, or administrative processes. In this regard, it shares many characteristics with other, nongeographic, information systems. GIS does present some problems peculiar to the geographic domain. Data management problems are much more complex because of the necessity of collecting, representing, and processing spatial data and their relationships. In an effort to solve problems arising from this complexity some have suggested that the fields of artificial intelligence and expert systems may offer assistance (Fisher et al 1988; Robinson and Frank 1987; Robinson et al 1986; Robinson et al 1988).

The field of Artificial Intelligence (AI) concerns the development of theories and techniques required for a computational engine to efficiently perceive, think, and act with intelligence in complex environments (Fox 1990). The areas of knowledge representation and search form two of the most important core concepts of this field. They are particularly relevant in addressing how to represent geographic information and find the solution to geographical problems.

A spinoff of AI known as Expert Systems (ES) is a field concerned with developing computer programs emulate the search behaviour of human experts in solving a problem. Closely related to ES is the field dealing with Knowledgebased Systems (KBS). KBS are systems that use domain knowledge to guide search in ways that differ from an expert's. The search and representation techniques are usually drawn from the AI-related research.

Classical ES/KBS systems have an inference engine and a knowledgebased. They are often developed using an AI-based language or development tool. In additions, there may be interfaces to relational database management systems and other system resources. Another approach is to embed the ES/KBS in an application-specific system. To the user it may appear as little has changed beyond improved performance and flexibility.

Expert/Knowledgebased Systems and GIS

Two general approaches to integrating GIS and ES/KBS technology have emerged recently. One is to build an application domain-specific KBS that sits separate from GIS but communicates with GIS. In this case, the ES/KBS uses the GIS as an information processing resource which produces data for the ES/KBS' use. Thus, little, or no, intelligence is added to the GIS, but rather the GIS becomes a component of an ES/KBS (e.g., White and Morse 1987).

Efforts such as the Intelligent Land Information Manager (ILIM) (Robinson and Zhang 1988), RESHELL (Goodenough et al 1987), Knowledgebased Spurious Polygon Processor (Robinson and Miller, 1989), and Intelligent Terrain Classification System (ITCS) (MacKay 1990), are to some extent directed towards the support of managing geographic information resources over time. Something a number of these approaches have in common is their "object-orientation" in handling geographic information. In this sense they are adding intelligence to geographic information systems.

The retrieval function of GIS is not limited to retrieving geographic information in response to a user query. It also includes the ability to interchange data with another GIS. In concept, the interchange function is analogous to handling the query from another system, not a human-user. Application of RESHELL involves communication and exchange of information among many expert systems, hence here is a system where not only is expertise in multiple domains present but also expertise on communicating among those domains. The Spatial Relations Acquisition Station (SRAS) addresses the problem of specifying and representing a spatial relation which is a fundamentally fuzzy concept. As such could be integrated with an existing query language such as the Structured Query Language.

Spatial analysis often included in the retrieval function, but is broken out here because there are special demands in the analysis of geographic data in response to an analytical query. The intelligent terrain classification system (ITCS) is an example of an application of ES/KBS in a geographic information processing domain. Although the basic geographic data describing the basic measurements of terrain are usually found in geographic databases, there is limited ability for the systems to automatically identify common geomorphic features. ITCS is an attempt to provide that capability while at the same time trying to formalize a model of glacial terrain features. In addition, this example illustrates how important the object-oriented paradigm is in development of intelligent geographic information processing capabilities.

Many people think of GIS as a map-making system. Indeed, one of the defining characteristics of many geographic endeavors is reliance upon, or production of, maps. The problems of name placement, map generalization, and others are among those most commonly identified as realizing the potential from integrating ES/KBS with GIS (Robinson et al 1988; Fisher et al 1988). In automating nautical charting, NOAA has engaged in developing an embedded ES/KBS which has knowledge of how to draw maps. This is also an example of an effort to build an embedded ES/KBS.

Trends in ES/KBS and GIS

The integration of ES/KBS and GIS is becoming increasing frequent, but not common. Their integration is still viewed, rightly or wrongly, as more of a research and development activity than for operational implementation. There are several trends developing that characterize the growing interrelationship of GIS and ES/KBS.

There is increasing interest in integrating commercially available expert system shell and GIS products. However, most efforts have concentrated on using ES/KBS techniques within GIS operations or using ES/KBS tools as an "interface" to GIS. In some ways, this may represent an effort to make the interface the GIS "expert."

AI languages are being used increasingly as tools to develop prototype systems. Often these efforts are directed towards developing specifications (e.g., Robinson and Zhang 1988; Roman 1986; Webster 1990). Others may be directed towards providing adjunct intelligence to GIS processes(e.g., Robinson and Miller 1989).

Increasingly there is use of AI techniques being incorporated in GIS. For example, Band (1989) describes the use of AI search and computer vision techniques to address problems of automatically extracting terrain features from digital elevation models. However, most efforts remain KBS rather than ES in that few attempts are made at formally capturing the expertise of some person who is expert in a particular geographic domain.

One of the most fundamentally important trends is in development of object-oriented approaches to GIS. It is quite natural for GIS users to manipulate geographic entities as objects. However, as Mohan and Kashyap (1988) have shown, there are extensions that need to be made to the object-oriented systems before the flexible manipulation of spatial information is possible. For example, a requirement of any object-oriented geographic information system is the ability to manage multiple inheritance.

References

Expert Systems and Knowledgebased Systems - General

Amble, T. 1987. **Logic Programming and Knowledge Engineering**, Addison-Wesley, Reading, MA.

Ballard, D.H. and C.M. Brown. 1982. **Computer Vision**. Prentice-Hall, Englewood Cliffs, NJ.

Bratko, I. 1986. **Prolog Programming for Artificial Intelligence**, Addison-Wesley, Reading, MA.

Buchanan, B.G. and E.H. Shortliffe. 1984.**Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**, Addison-Wesley, Reading, MA.

Dabrowski, C.E., E.N. Fong, and D. Yang. 1990. *Object Database Management Systems: Concepts and Features*, **NIST Special Publication 500-179**, U.S. Government Printing Office.

Ford, Nigel. 1987. **How Machines Think: A General Introduction to Artificial Intelligence Illustrated in Prolog**, John Wiley & Sons, New York.

Harmon, P., R. Maus, and W. Morrissey. 1988. **Expert Systems Tools and Applications**, John Wiley & Sons, Inc., New York, pp. 289.

Jackson, P.C. 1985. **Introduction to Artificial Intelligence**, Second Enlarged Edition, Dover Publications, New York.

Mylopoulos, J. and M.L. Brodie (editors). 1989. **Readings in Artificial Intelligence & Databases**, Morgan Kaufmann, San Mateo, CA.

Reddy, 1988. *Foundations and Grand Challenges of Artificial Intelligence*, AI Magazine, 9(4):9-23.

Rich, E. 1983. **Artificial Intelligence**, McGraw-Hill, New York.

Winston, P.H. 1984. **Artificial Intelligence**, Second Edition, Addison Wesley, Reading, MA.

Winston, P.H. and B.K.P. Horn. 1981. **Lisp**, Addison-Wesley, Reading, Mass.

Zodnik, S.B. and D. Maier (editors). 1990. **Readings in Object-Oriented Database Systems**, Morgan Kaufmann, San Mateo, CA.

Geographic Information Systems and ES/KBS

Band, L. 1989. *Automating Topographic and Ecounit Extraction from Mountainous Forested Watershed*, **AI Applications in Natural Resource Management**, 3(4): 1-11.

Coughlan, J.C. and S.W. Running. 1989. *An Expert System to Aggregate Biophysical Attributes of a Forested Landscape within a Geographic Information System*, **AI Applications in Natural Resource Management**, 3(4): 35-43.

Davis, J.R., J.R.L. Hoare, and P.M. Nanninga. 1986. *Developing a Fire Management Expert System for Kakadu National Park, Australia*, **Journal of Environmental Management**, 22:215-227.

DeMers, M.N. 1989. *Knowledge Acquisition for GIS Automation of the SCS LESA Model: An Empirical Study*, **AI Applications in Natural Resource Management**, 3(4): 12-22.

Fisher, P.F., W.A. Mackaness, G. Peacegood and G.G. Wilkinson. 1988. *Artificial intelligence and expert systems in geodata processing*, **Progress in Physical Geography**, 12(3):371-388.

Fullford, B. and V.B. Robinson. 1987. *Development of a Prototype Rule-Based System for Evaluating Agricultural Land Suitability: A Progress Report*, **Proceedings URISA** (Alberta Chapter), Calgary, AB, pp. 11-28.

Goodenough, D.G., M. Goldberg, G. Plunkett, and J. Zelek. 1987. *An Expert System for Remote Sensing*, **IEEE Transactions on Geoscience and Remote Sensing**, GE-25(3): 349-359.

Hanson, A.R. and E.M. Riseman. 1978. *VISIONS: A Computer System for Interpreting Scenes*, pp. 203-333, in **Computer Vision Systems**, A.R. Hanson and E.M. Riseman, Eds., Academic Press, New York, New York.

Havens, W. and A. Mackworth. 1983. *Representing Knowledge of the Visual World*, **Computer**, 90-96.

Hepner, G.F. and T. Logan, N. Ritter, and N. Bryant. 1990. *Artificial Neural Network Classification Using a Minimal Training Set: Comparison to Conventional Supervised Classification*, **Photogrammetric Engineering and Remote Sensing**, 56(4):469-473.

Lundberg, G. and V.B. Robinson. 1988. *Computers That Know: A Tutorial*, **Computers, Environment, and Urban Systems**, 12(1):49-72.

MacKay, D.S. 1990 in press. **Intelligent Hillslope Classification System**, Technical Report 90/3, Institute for Land Information Management, University of Toronto, Erindale Campus, Mississauga, Ontario.

McKeown, D.M., W.A. Harvey, and J. McDermott. 1985. *Rule-based Interpretation of Aerial Imagery*, **IEEE Trans. Pattern Analysis and Machine Intelligence**, 7(5):570-585.

Mohan, L. and R.L. Kashyap. 1988. *An Object-Oriented Knowledge Representation for Spatial Information*, **IEEE Trans. on Software Engineering**, 14(5):675-681.

Orenstein, J.A. and F.A. Manola. 1988. *PROBE Spatial Data Modelling and Query Processing in an Image Database Application*, **IEEE Trans. on Software Engineering**, 14(5):611-629.

Perkins, W.A., T.J. Laffey, and T.A. Nguyen. 1986. *Rule-based Interpretation of Aerial Photographs Using the Lockheed Expert System*, **Optical Engineering**, 25:356-362.

Robinson, V.B. 1990 in press. *Interactive Machine Acquisition of a Fuzzy Spatial Relation*, **Computers and Geosciences**.

Robinson, V.R. and R. Miller. 1989. *Intelligent Polygon Overlay Processing for Natural Resource Inventory Maintenance: Prototyping a Knowledge-based Spurious Polygon Processor*, **AI Applications in Natural Resources Management**, 3(4):1-12.

Robinson, V.B. and G. Zhang. 1988. *Modelling Consistency-Preserving Land Database Transactions in a Logic Programming Environment*, **Proceedings** GIS/LIS '88, San Antonio, Texas, USA.

Robinson, V.B. 1988. *Implications of Fuzzy Set Theory Applied to Geographic Databases*, **Computers, Environment, and Urban Systems**, 12(2):89-98.

Robinson, V.B., A.U. Frank, and M.A. Blaze. 1988. *Expert Systems and Geographic Information Systems: Review and Prospects*, in D.T. Pham (editor), **Expert Systems in Engineering**, AI in Industry Series, IFS (Publications) Ltd., Kempston, Bedford, England, pp. 203-213.

Robinson, V.B. and A.U. Frank. 1987. *Expert Systems for Geographic Information Systems*, **Photogrammetric Engineering and Remote Sensing**, 53(10):1435-1441.

Robinson, V.B. and R. Wong. 1987. *Acquiring Approximate Representations of Some Spatial Relations*, **Proceedings** Eighth International Symposium on Computer-Assisted Cartography, Baltimore, Maryland, pp. 604-622.

Robinson, V.B., A.U. Frank and M. Blaze. 1986. *An Introduction to Expert Systems for Land Information Systems*, **Journal of Surveying Engineering**, 112(2): 109-118.

Robinson, V.B., A.U. Frank and M. Blaze. 1986. *Expert Systems and Geographic Information Systems: Review and Prospects*, **Journal of Surveying Engineering**, 112(2): 119-130.

Robinson, V.B., M. Blaze, and D. Thongs. 1986. *Man-Machine Interaction for Acquisition of Spatial Relations as Natural Language Concepts*, in **Geographic Information Systems in Government**, B.K. Optiz (ed), A. Deepak Publishing, Hampton, VA, pp. 433-454.

Robinson, V.B., D. Thongs, and M. Blaze. 1986. *Representation and Acquisition of a Natural Language Relation for Spatial Information Retrieval*, **Proceedings**, Second International Symposium on Spatial Data Handling, Univ. of Washington, Seattle, WA, pp.472-487.

Robinson, V.B. and A.U. Frank. 1985. *About Different Kinds of Uncertainty in Geographic Information Systems*, **Proceedings**, Seventh International Symposium on Computer-Assisted Cartography, Washington, D.C., pp. 440-450.

Roman, G. 1986. *Formal Specification of Geographic Data Processing Requirements*, **Proceedings** International Conference on Data Engineering, IEEE, pp. 434-446.

Tailor, A., A. Cross, D.C. Hogg, and D.C. Mason. 1986. *Knowledge-based Interpretation of Remotely Sensed Images*, **Image and Vision Computing**, 4(2):67-83.

Webster, C. 1990. *Rule-based Spatial Search*, **International Journal of Geographical Information Systems**, 4(3): 241-260.

White, W.B. and B.W. Morse. 1987. *ASPENEX: An Expert System Interface to a Geographic Information System for Aspen Management*, **AI Applications in Natural Resources Management**, 1:49-53.

GIS/EXPERT SYSTEM DEMONSTRATION:
NOAA's CartoAssociate

Steve Luckey, Artificial Intelligence Group
Nautical Charting R&D Laboratory
Nautical Charting Division/Coast and Geodetic Survey
National Oceanic and Atmospheric Administration
Rockville, Maryland 20852

Objective

This presentation demonstrates the current status of a prototype system under development by the National Ocean Service, Coast and Geodetic Survey that utilizes object-oriented and expert systems techniques to aid in the automated production of NOAA's nautical navigation charts. After describing the basics of the prototype, this paper discusses the menu selection sequence and the resulting effects that were shown at the live demonstration. References to the display screen user interface menu item selections are denoted by enclosed single quotes.

System Platform

The hardware platform is a Tektronix 4317 engineering workstation with a 19" color screen, 8 MB RAM memory, and 240 MB disk space running under Unix. The software packages are: the Analyst Information Management System and Humble Expert System Shell, both from Xerox Corporation, Servio Corporation's GemStone Object Data Base Management System, and Tektronix Corporation's full implementation of the Smalltalk-80 language and environment.

Description of Software System

In this system the software engineer has available the development tools provided by the Analyst, Smalltalk, and Humble integrated environment. This includes facilities for data management, data base creation, editing, creation of maps with scaling algorithms and links to data bases, and the integrated link to the expert system shell. With the windowing capabilities provided by Analyst and Smalltalk, the programmer can bring up windows on the database, the Humble Rule Editor, the Smalltalk environment with full source code implementation, and if desired, a window on a user created map or chart to display results graphically.

These windows can be displayed singly or concurrently, and by clicking the mouse inside the appropriate window, activate the window. The programmer can then alternate selections between windows, changing data base attributes, adding or changing values, writing or changing Humble rules, writing Smalltalk code, and displaying the results at the same time.

The CartoAssociate Project

This project is a beginning effort to automate the more difficult aspects of chart-making now performed in the manual mode. Design details related to these aspects are described in [Pendleton, 1991] and [Luckey, 1988]. The Galveston Harbor, Texas, area was selected as a test site for the prototype because this area has overlapping charts at different scales allowing us to compare our results to existing charts. We decided to select an area roughly centered on Pelican Island containing typical aspects of the harbor/nautical channel chart depiction problem .

In creating a suitable user interface, we started with a blank chart window at a specific scale, centered on a specific spot in the western hemisphere, with the image on the screen as large as possible. In order to retain all the menu-driven mapping functions already present in the Analyst, along with the capability to write our own functions, Xerox supplied additional Smalltalk code allowing us to zoom in on a section of the Continental United States Map (already in existence within Analyst) and create full-screen chart windows at any required scale.

We have added nonmodal user interface features to the existing Analyst menu functions, including a chart compilation selection. This section leads into a sub-menu for further selection of specific NOAA nautical chart compilation activities. The interface is nonmodal like the Smalltalk-80 environment, in that all facilities are simultaneously available to the user; e.g., it is not necessary to first exit a data retrieval mode in order to enter a chart editor mode.

The Data Base

The CartoAssociate's design concept requires that the data drives the system, that is, the system looks first to the data base for answers and values. Once data values are entered into the data base and validated, they are considered accurate and are not modifiable by any subsequent compilation process. The Analyst provides a feature whereby one can easily create and test relational-type data bases with cartographic attributes and values for each type of chart feature. Although not an object-type data base, it is a quick way to build and maintain a test data base for prototype development. The next project task is to fully load Servio Corporations's object data base management system GemStone with chart feature data to transition the prototype to a full feature object data base.

Our data bases contain attributes and values for real-world features appearing on NOAA's nautical charts. These include: shipping channel specifications and channel segments; shorelines and dangers, including rocks, wrecks, and pilings; navigational aids, including buoys and range lights; and text data for buoy labels.

The Expert System

By integrating the Humble expert system shell into the Analyst/Smalltalk environment, we have added object "intelligence" to an already rich programming paradigm that can assist in solving very complex chart compilation problems. One of these problems is to be able to determine the appropriate icon symbols to represent real-world objects on a chart. This must be done within an overall context of detecting and resolving conflicts between competing features of the same or different types for the same chart "white" space.

The knowledge base rule set was created such that the rule parameters were requirements to be satisfied before the expert system could derrive a solution. One simple way to meet these requirements and satisfy our data-driven restraint was to make the expert system parameters a subset of the data base attributes for that entity type. These are the entities about which the CartoAssociate makes inferences. A set of parameters was devised to characterize the data values for instances of danger features, such as their geographic positions, height above mean sea level, color, flashing rate, name, etc. A Dangers Knowledge Base of "if--then" rules was then created in Humble rule format, as shown in Figure 1, and identified as an abstract data type called "Dangers" to represent the set of real-world hazards to navigation such as rocks, wrecks, pilings and platforms.

From within the CartoAssociate's Smalltalk code: (1) the data base is queried; (2) a feature object is interrogated for its attribute/parameter values; (3) these values are passed to the Dangers Knowledge Base; (4) and the inference engine returns an answer. The Humble inference engine processes the rules to infer an answer/value which, for this particular request, is used as the key to a table of icons containing the selected symbol. In this manner, when given appropriate parameter values about a danger object from the data base, the CartoAssociate's backward-chaining rule processing mechanism proceeds to search the space of feasible possibilities to generate a correct icon symbol selection.

Channel Representation

Shipping lanes, or channels, are important to the mariner and play a very important role in NOAA's chart-making process. Channels have specific rules about how they are displayed and how other related nearby features are depicted.

Channel specification databases were built using data from Corps of Engineers blueprints to capture channel specific data including channel name, channel azimuth, channel width, and the channel turning points.

Smalltalk code was created to address this data base, extract the latitude and longitude for the channel turning points, convert the points to screen coordinates, compute and connect the intercept points between adjacent channel lines, and display the results as a series of lines representing real-world navigational channels.

A great deal of effort went into acquiring the Corps of Engineers data, building the data bases, and deriving the code to compute and connect the channel segment intercept points. This was chosen as a good starting point in the development because channel displays should be considered "super features" of the nautical chart. Such entities will be implemented as compound objects, i.e., features composed of other more basic nested features, in the object data base as the prototype's implementation proceeds. Currently, the prototype's "intelligence" is limited to the resolution of conflicts with individual chart point features. Compound feature resolution will be attacked as a second step.

Icon Conflict Resolution

A primary item of interest is a navigational aid called "buoy". Buoys are normally channel markers and, as such, are usually associated with and near to channels. On the Atlantic side of the United States, the red buoys mark the right side of the channel and the green buoys mark the left side as the ship enters the channel from the ocean side. This will become important at a later stage when the CartoAssociate will be required to resolve certain conflicts between overlapping features and labels in areas of the chart near channel boundaries.

Icons representing real-world objects, such as buoys, appearing in the Nations's waterways must meet certain requirements for portrayal on NOAA nautical charts, particularly when these objects define channels, establish right-of-way, or signify a danger. Whenever an icon overlaps or conflicts with another feature, there are certain rules and specifications that the cartographer must follow for moving, removing, or selecting an alternate portrayal. Under the chart compilation model, each chart object knows about itself and its own behavior (i.e., how it can portray itself) under a variety of circumstances. This behavior is defined by software routines and knowledge base rules when the object is initially defined.

With this in mind, the design approach to the icon conflict resolution problem was to think of each icon object as having an x, y grid coordinate system with an origin located at its center, thus dividing the small potential conflict area about the object into four quadrants. Close neighbors are then tested for icon overlap and, if found, the overlapping object and any other close neighbors are noted as occupying their respective quadrants. If a quadrant is occupied by more than one neighbor, the closest one is noted. These quadrant occupations become parameter values for rules in the knowledge base as used by the CartoAssociate to choose an unoccupied quadrant.

Utilizing this knowledge, together with distance information between object and neighbor, the CartoAssociate determines a distance and direction to shift the icon symbol. If all quadrants are occupied, then a system solution is not possible and the icon is added to a deferred icon collection for placement via user interaction, after all system performed icon placements are finished.

Label Conflict Resolution

One of the heuristics associated with chart features and channels is that chart features and/or their accompanying labels cannot be placed in the interior of a channel graphic. To satisfy this heuristic in terms of buoy labels, a Buoy Label Knowledge Base was created to deal with the rule parameters of channel width and azimuth, buoy color, and the specific buoy label characteristics that define the buoy label.

As before, the appropriate feature database is interrogated, attributes/parameter values are extracted, and the values are passed to the Buoy Label Knowledge Base for resolution. After chaining through the knowledge base, the inference engine returns a set of appropriate locations for each characteristic of the label. These become the starting candidate locations for the label. Initial chandidate positions are established according to the 8 candidate box positions suggested in [Yoeli, 1972] and [Imhof, 1975] with a revised priority ordering that meets NOAA nautical charting requirements.

The candidate locations are constructed as bounding boxes which are tested against the bounding boxes of the neighbor objects. Since the candidate boxes are in priority order in the tested array, the first candidate box that passes the test for no neighbor conflicts is the one selected. If a candidate box cannot be found to pass the test, the label is added to a deferred label collection for placement via user interaction, after all system performed label placements are finished.

Display

Feature icons and labels are displayed after all conflicts are either resolved or deferred. Several interactive user interface display tools, involving the use of the screen cursor and mouse, have been developed to aid the user in placing deferred labels and icons.

Figure 2 shows a display of the chart feature data as icons before any expert system processing is performed. It can be seen that icons for piles, channels, and other features as well as labels for text are in conflict in numerous locations on the display. Figure 3 shows the results after expert system processing in which these conflicts and overlaps have been resolved.

Future Development

As development contines, we plan to focus on three key issues.

1. Add the object database component, Servio Corporation's GemStone Data Base Management System, to the prototype and populate it for a series of data base volume tests. This will be crucial to future work with more complex charting data.

2. Expand the user interface by creating a more natural entry into the system and then streamlining and standardizing the functionality within the compilation code development.

3. Continue to expand the prototype's functionality in conflict resolution operations and other fundamental cartographic problems using expert systems, GIS concepts, and the inherent power within the object environment to manage the complexity of the evolving software system.

References

Imhof, E., 1975. *Positioning Names On Maps.* American Cartographer 2(2), 128-144.

Luckey, Stephen E, and Pendleton, Dave L., *Object-Oriented Concepts As Applied To Automated Chart Compilation.* Proceedings of the U.S. Hydrographic Conference '88, 171-177, April, 1988.

Pendleton, Dave L. *NOAA's Experience With Object Programming/Data Bases and Expert Systems.* Proceedings of the NIST GIS Standards Laboratory Workshop on Technology Integration (this volume), 1991.

Yoeli, P., 1972. *The Logic Of Automated Map Placement.* Cartograph. J. 9(2) 2(2), 99-108.

An Example Rule Set in the Chart Editor

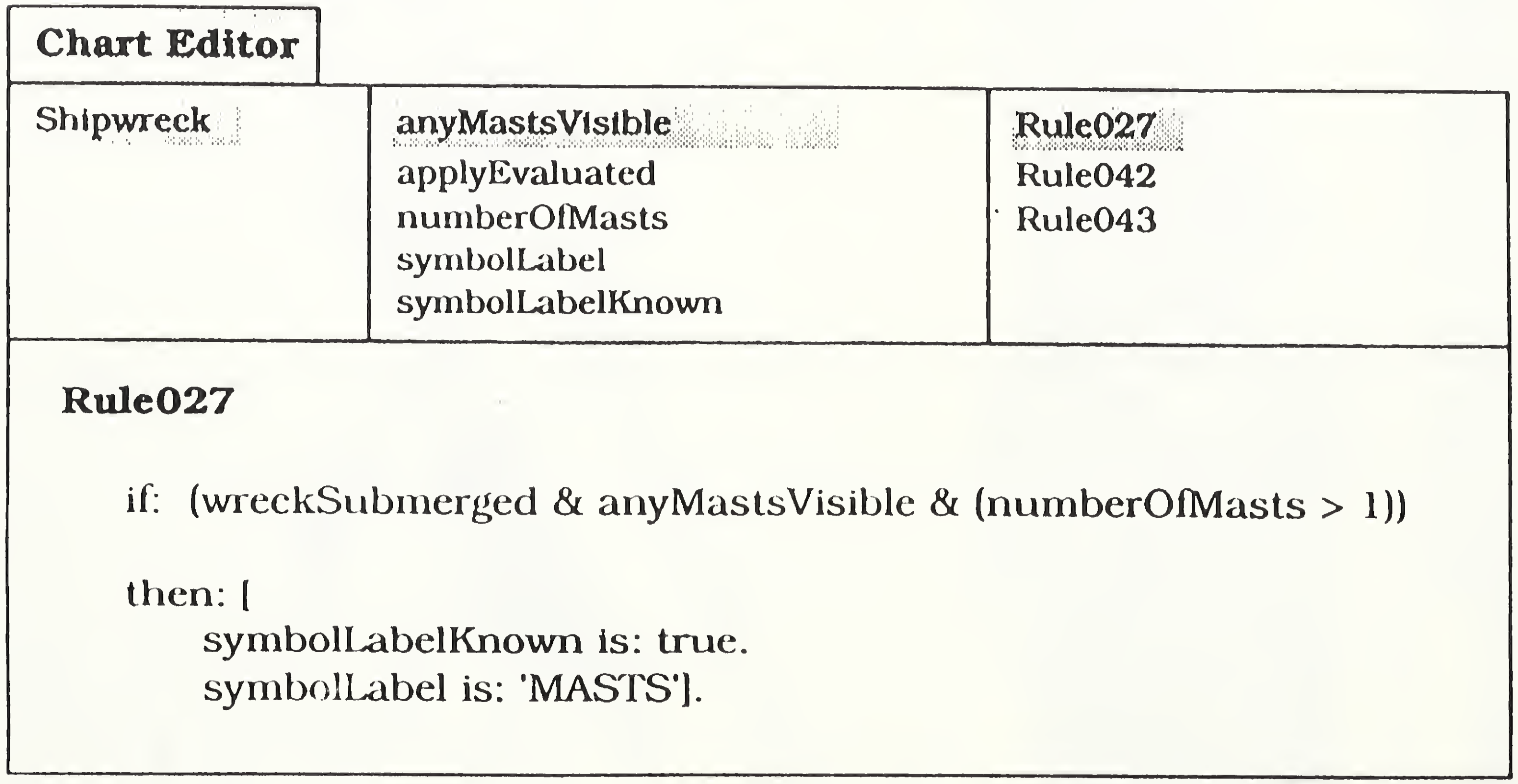


Chart Editor

```
Shipwreck          anyMastsVisible          Rule027
                   applyEvaluated           Rule042
                   numberOfMasts            Rule043
                   symbolLabel
                   symbolLabelKnown
```

Rule027

```
if:  (wreckSubmerged & anyMastsVisible & (numberOfMasts > 1))

then: [
     symbolLabelKnown is: true.
     symbolLabel is: 'MASTS'].
```

Figure 1. The Humble Expert System Shell's Chart Editor

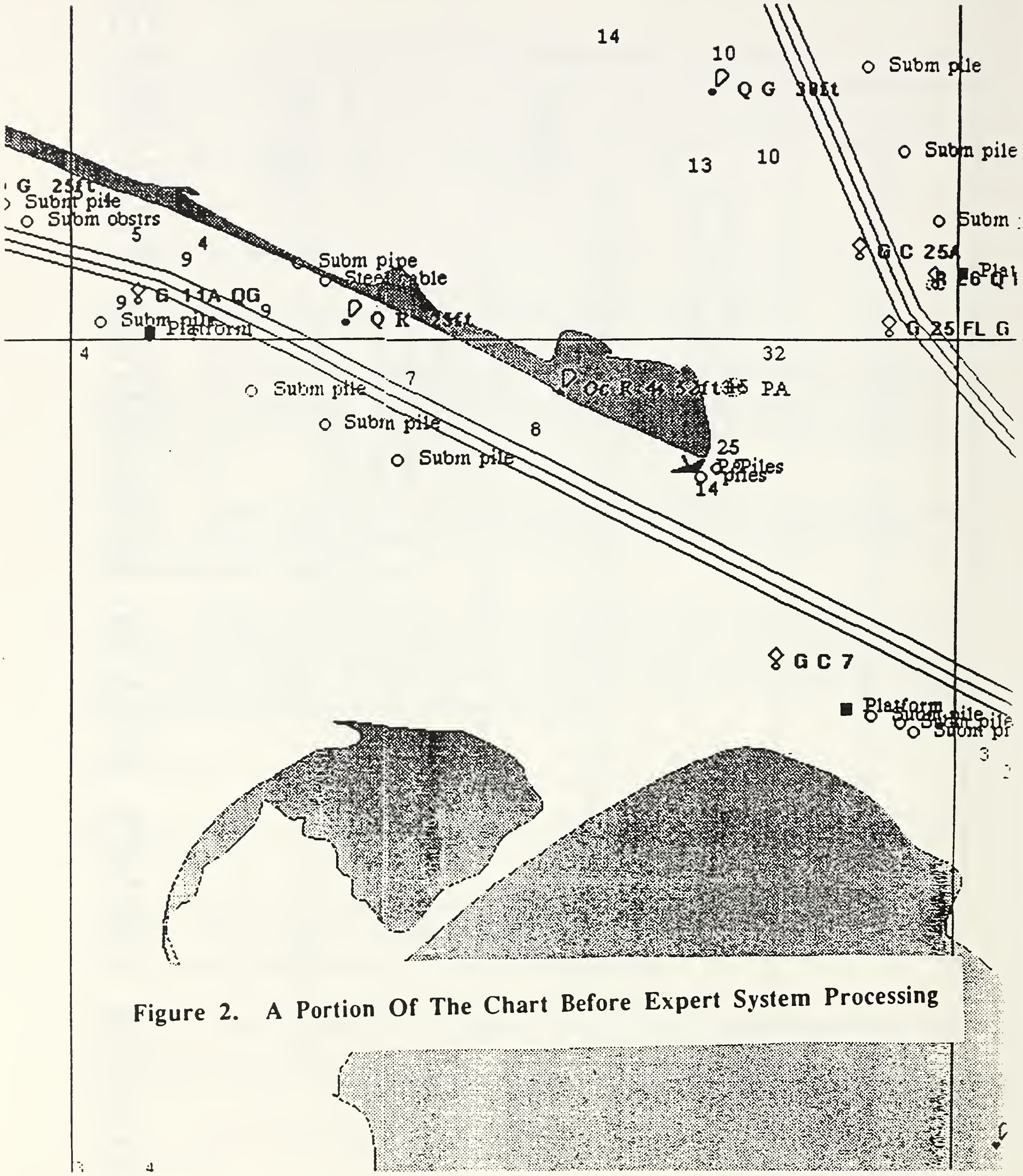


Figure 2. A Portion Of The Chart Before Expert System Processing

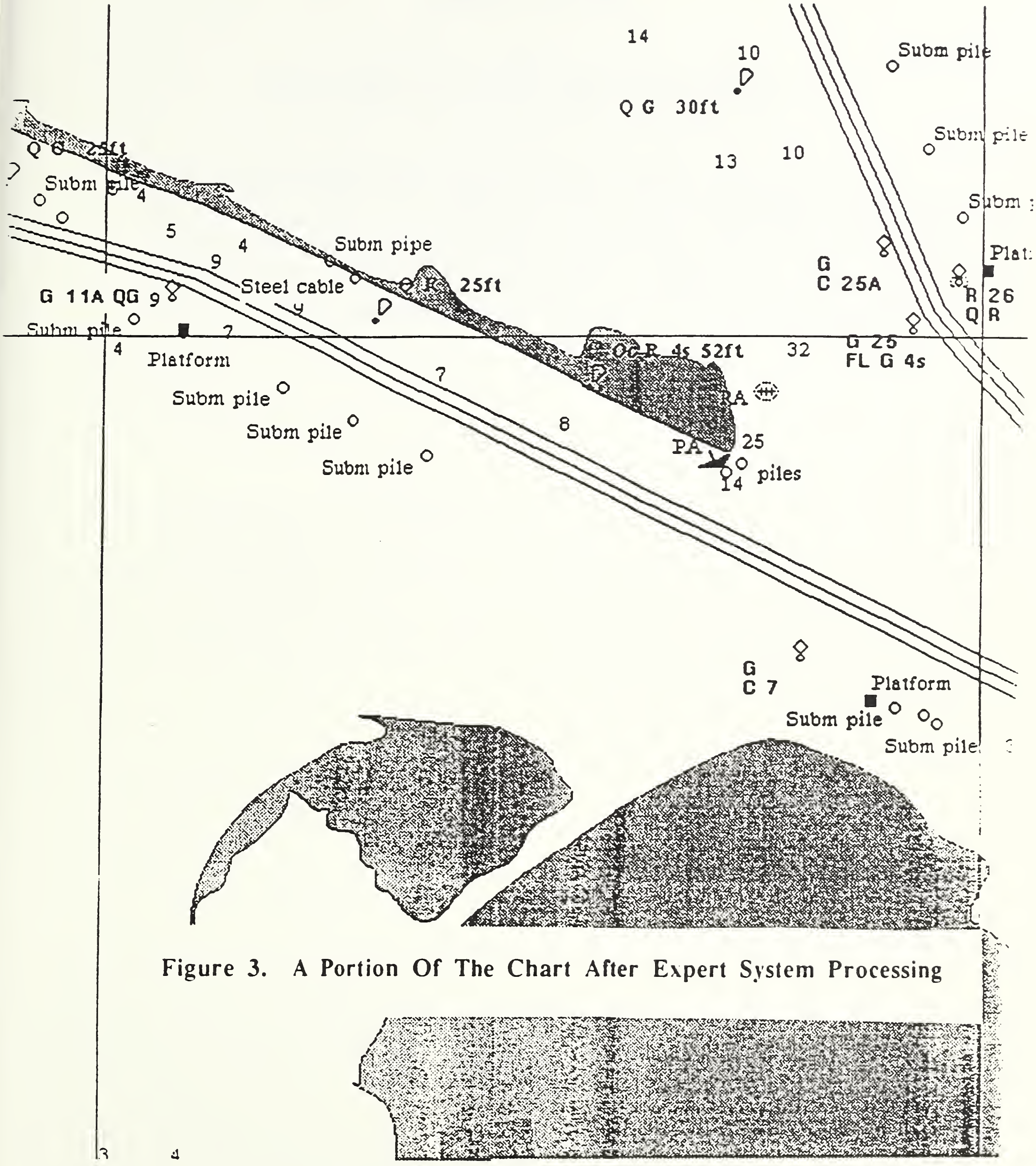


Figure 3. A Portion Of The Chart After Expert System Processing

OBJECT CONCEPTS

by

Christopher Dabrowski and Elizabeth Fong

This paper provides a description of the concepts generally associated with the object-oriented programming paradigm. Object-oriented programming is a powerful tool for developing computer systems that represent and process complex structural information. This paper provides a tutorial introduction to object concepts and describes how the object-oriented approach may be used in modeling real-world information systems.

1.0 BACKGROUND

This paper provides an introduction to the concepts that underlie the object-oriented programming paradigm and illustrates how these concepts can be applied. Object-oriented programming is a powerful tool for developing computer systems that represent and process complex, structural information. For many applications, the object-oriented approach offers significant advantages over conventional software by providing clearer structure, reusability, and easy maintainability.

1.1 Brief History

The origins of object-oriented programming lie in experimental computer languages developed in the 1960s, such as Simula [DAHL66] and CLU [LISK77]. The first programming language that exhibited all of the major characteristics of the object paradigm was Smalltalk [GOLD83]. Smalltalk resulted from a research effort undertaken in the 1970s at the Xerox Palo Alto Research Center. The term "object-oriented" was coined during the course of the development of Smalltalk [HORO83].

Subsequently, other object-oriented programming languages have emerged including the Flavors subsystem in ZetaLisp [CANN82], C++ [STRO86], and the Common Lisp Object System [CLOS88]. This has been followed by the introduction of commercial object-oriented database management systems including Iris [FISH89], Zeitgeist [FORD88], GemStone [MAIE87], and ORION [KIM89]. Object-oriented database management systems will be described in subsequent papers in this publication.

1.2 Organization Of This Paper

The organization of this paper follows the concepts of the object paradigm. Section 2 introduces objects as fundamental computational entities. Section 3 discusses object classes, or "types" of objects. In Section 4, the ability to create class hierarchies that share class definitions through inheritance is discussed. Sections 5 and 6 introduce additional aspects of the object paradigm. Section 7 presents conclusions.

Before proceeding, it is appropriate to state that there is no single agreed upon definition of what the terms "object" and "object-oriented" mean. Recently, there have been several efforts to standardize on object systems including [OODB90]. As a result, there has been a growing trend to use the term "object" instead of "object-oriented."[1]

[1]In this paper, the term "oriented" will be omitted except referring to object-oriented programming languages or object-oriented programs.

2.0 OBJECTS

Conceptually, an object is something that is perceived as an identifiable, self-contained unit that can be distinguished from its surroundings. As such, an object has essential information about itself and may be able to perform operations that cause it to exhibit a behavior. In the real world, we can easily identify many objects. An example might be an airplane. An airplane is readily identifiable and is distinguishable from its surroundings. An airplane can be said to maintain certain information about itself; e.g., its color, weight, and altitude. An airplane can also perform certain operations that cause it to exhibit a behavior; e.g., it can fly.

In a computer program, the concept of an object is implemented as a software component. This component consists of both attributes[2] that contain important information about the object and procedures, called methods, that define the object's behavior.

- **A Software Component That Contains Information And Procedures**

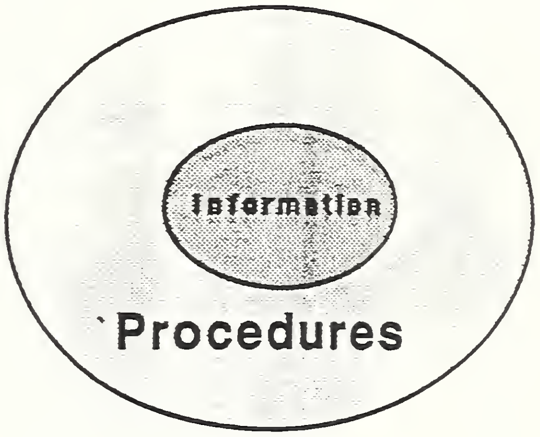


Figure 1. An Object

---

[2]In many object-oriented languages the term "instance variable" is used instead of attribute. However, attribute is more widely used in the data processing community.

A particular computer application may contain many objects. Each object has an unique identifier known as an <u>object identifier</u>, or <u>handle</u>. The unique identifier serves to distinguish objects from each other. Each object also has a <u>state</u> that is defined by the value of its attributes. During the execution of a program, the object's state may change; e.g., the value of its attribute values change. However the object's identifier remains fixed.

The function of objects in a computer program is to model real-world objects by capturing their most important informational and behavioral aspects. As such, objects in a computer program are said to represent <u>abstractions</u> of real-world objects.

An airplane can be modeled as an object. An airplane object can have a set of attributes that constitutes an internal state which can change over time. The attributes for the airplane object might be Altitude, Groundspeed, and Fuel. The airplane object's behavior may be represented by a set of methods that describe how the airplane takes off, climbs, and descends.

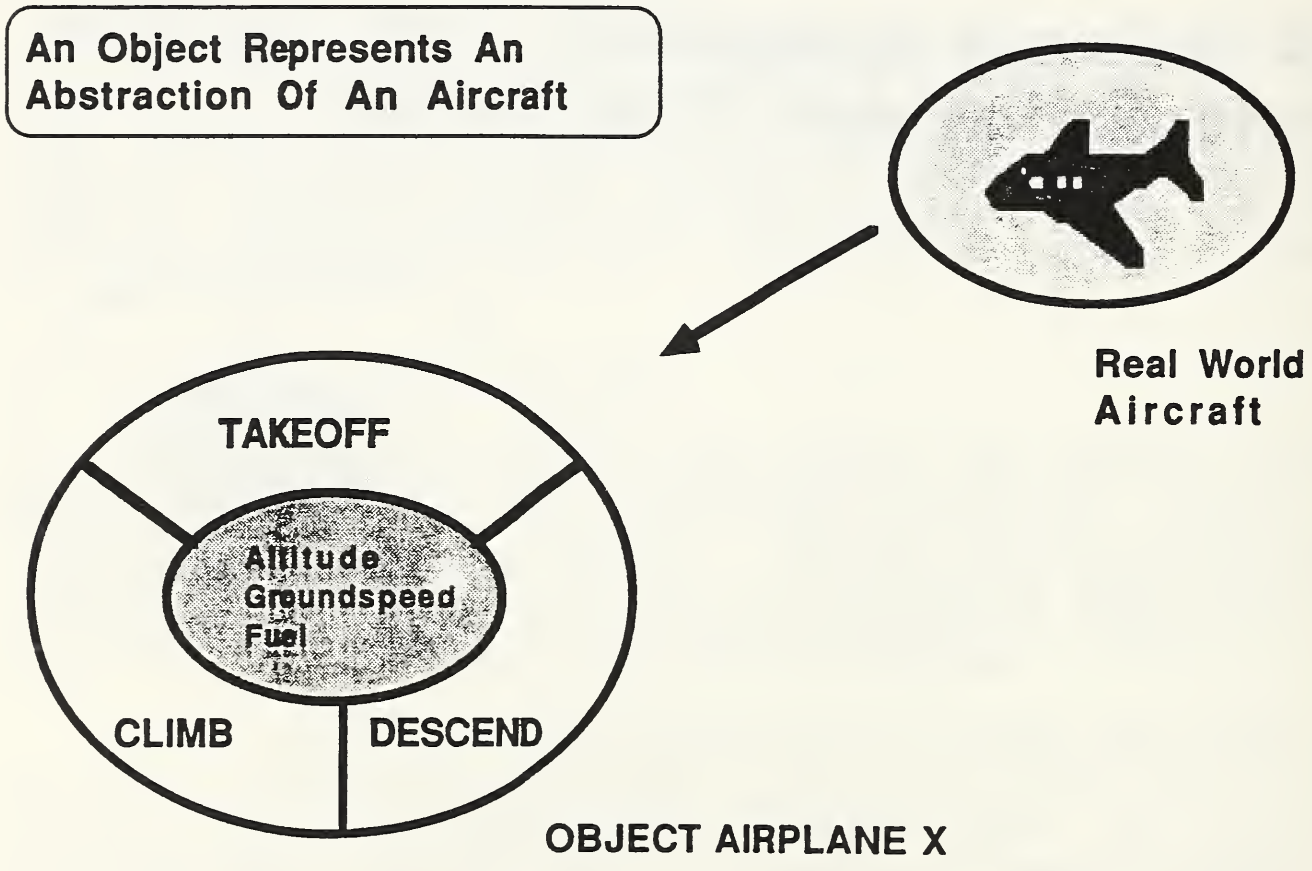


Figure 2. Modeling An Aircraft As An Object

The ability to represent abstractions of real-world objects, events, systems, and processes is one of the most important aspects of object-oriented programming languages.

2.1 Communicating With Objects: Messages and Methods

Communication with an object is accomplished by sending a message to it. A message is a request made to an object to perform an action or carry out an operation. Messages contain the name of the message, the object identifier of the object to which the message is being sent, and any arguments that may be necessary to perform the operation. When an object receives a message, it invokes one of its methods to perform the operation. A method may itself send messages to other objects.

An example illustrates how message passing works. A computer program simulating the flight of airplanes may need to model the descent of an aircraft. This is done by sending a DESCEND message to an AIRCRAFT object. In this case, the message contains the object identifier of the airplane object and a single argument. The argument contains the altitude to which the aircraft must descend - 10000 feet. When the AIRCRAFT object receives the message, the DESCEND method is invoked. The method performs the descend operation. During this operation, the value of the Altitude attribute is changed to 10000, thus altering the object's internal state.

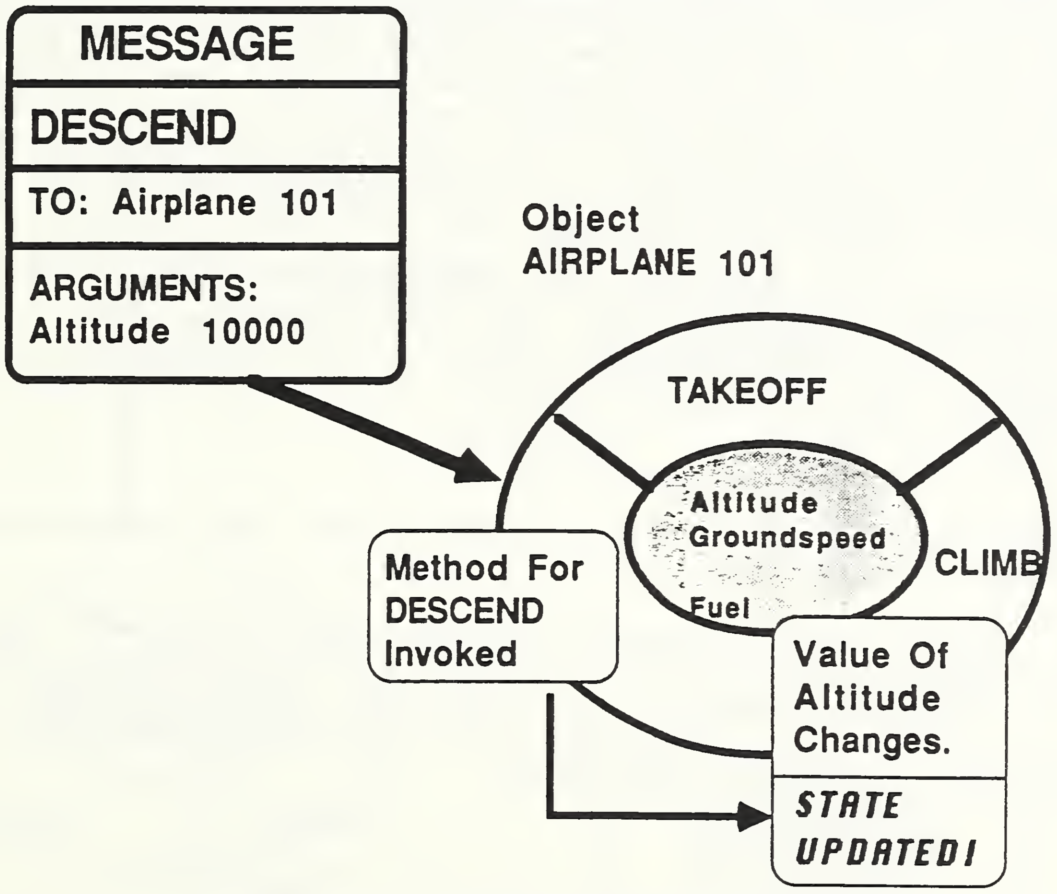


Figure 3. Modeling Aircraft Descent Through Message Passing

In an object system, each object may respond to many different messages. The entire set of messages an object responds to constitutes its public interface and is sometimes referred to as a protocol.

2.2 Encapsulation of Objects

An object's internal state is accessible only by the object itself. The values of the object's attributes cannot be accessed and changed directly by any external agent. Even the names of the object's attributes are not known to external agents. Similarly, the details of the operations performed by the object's methods are not visible to any other external agent, including the sender of the message. This characteristic hiding of the object's internal state is known as object encapsulation.

An object's internal state can be effected only indirectly through message passing and method invocation. When a method is invoked, only the result of the operation may be seen by external agents. Intuitively, encapsulation of an object makes it into a "black box."

When the AIRCRAFT object receives the DESCEND message, the details of the method performing the descend computation are not seen. Similarly, the change to the object's internal state is also unseen. An external agent that wishes to know the new altitude of the AIRCRAFT object must use the object's protocol. To provide the value of the Altitude attribute, the protocol can be defined to include a SHOW-ALTITUDE message. A corresponding method can also be defined that returns the value of the Altitude attribute.

To summarize, an object can be effectively divided into two parts: an interface part and an implementation part. The interface part is the object's protocol, or the messages to which it will respond. The implementation part is the object's internal state and its methods.

2.3 Contrasting Object-Oriented Programming With Conventional Programming

In an object system, message passing between objects is the predominant mode of computation. In contrast, conventional program organization is characterized by program control through subroutine invocation. The differences between the programming paradigms is illustrated in the following figures.

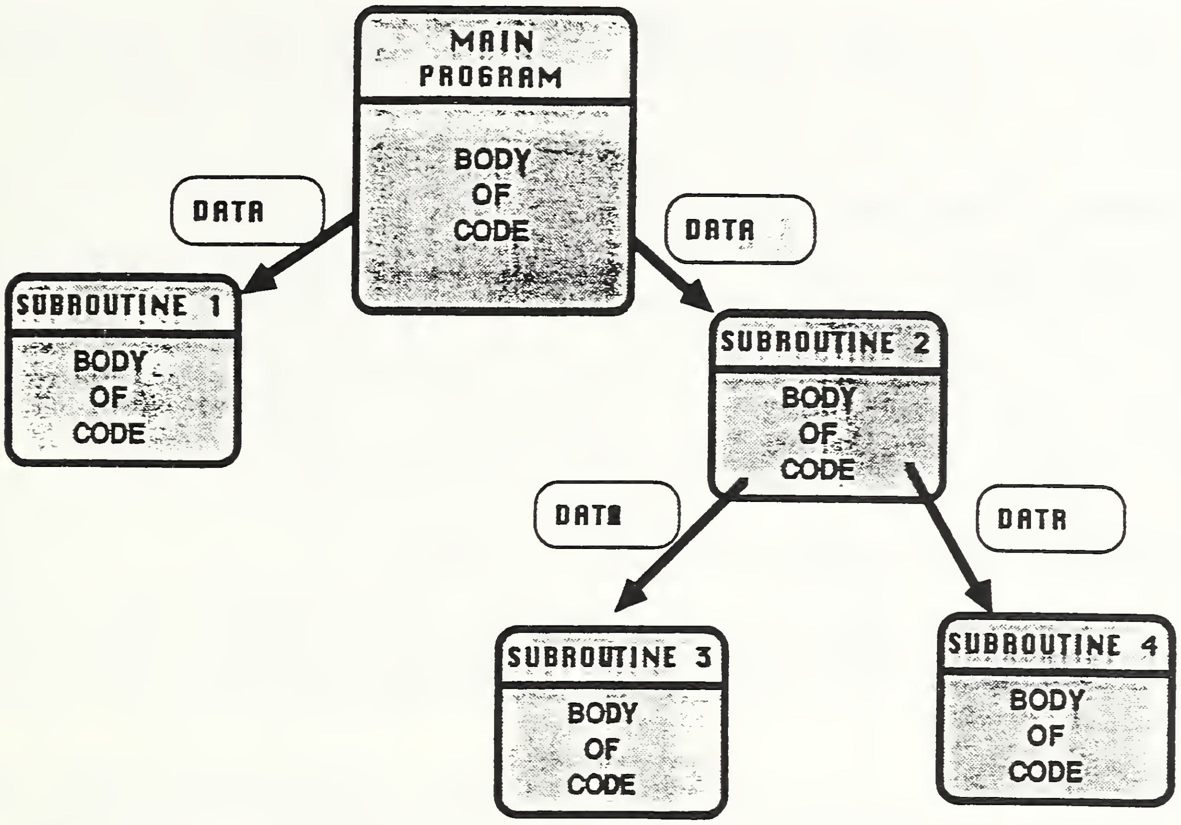


Figure 4. Flow Of Control In Conventional Software

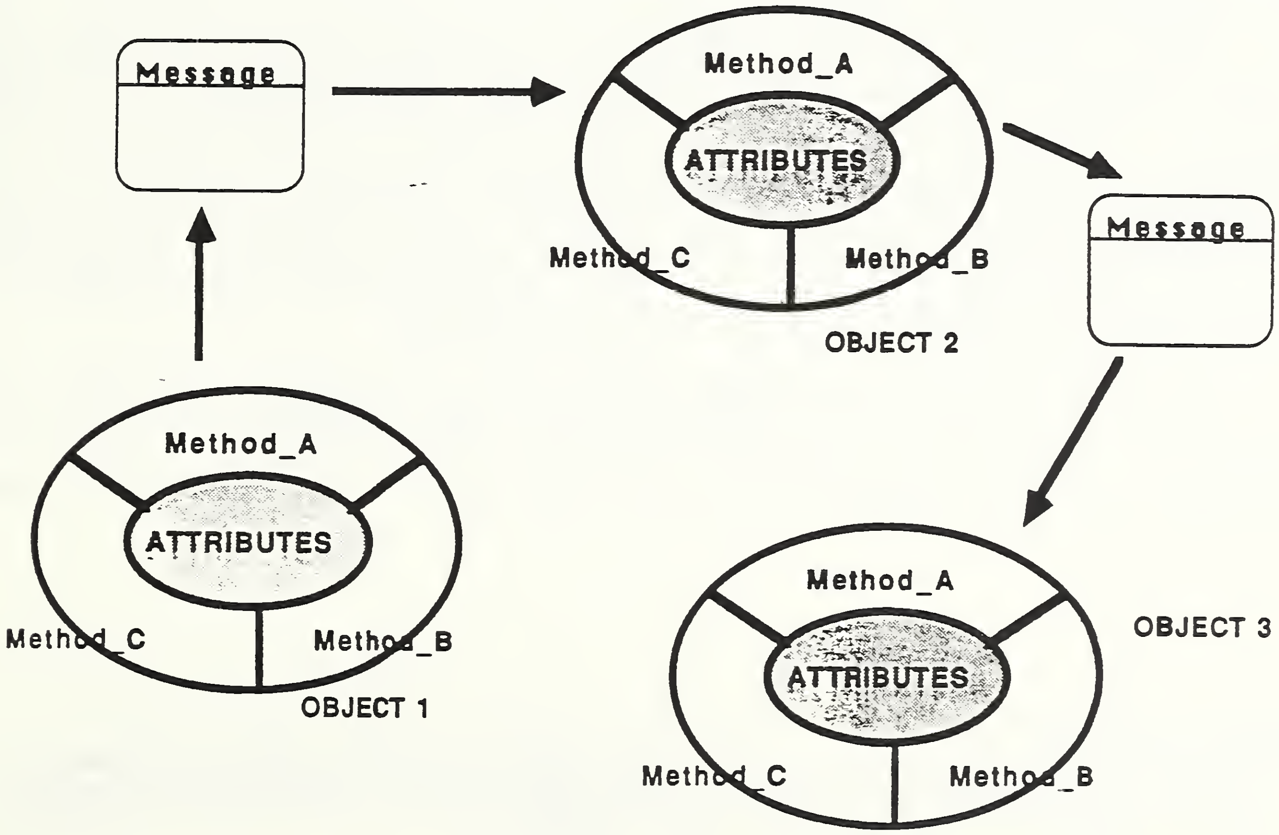


Figure 5. Flow Of Control In An Object-Oriented Program

These characterizations, however, are not absolute. In many programming languages, message passing can be mixed with conventional procedural control, as is the case with CLOS and C++.

3.0 CLASSES OF OBJECTS

Classes define kinds, or types, of objects. A class definition is a description consisting of a set of attributes that all objects of the class will have, a set of messages that these objects will respond to, and a set of methods for implementing operations associated with the messages.[3] Classes are also sometimes referred to as abstract data types [JOSE89].

3.1 Creating Objects From Class Definitions

In a computer program, many classes may be defined. Each class description serves as a template from which new objects are created. These objects are called instances of the class they belong to.

Every object is an instance of a class. Each object has the same attributes provided in the class description. Each object responds to the same messages and implements the same operations defined by the methods of the object's class.

3.2 Relationships Between Classes

Class definitions can describe real-world relationships that link dissimilar kinds of objects. For instance, in an actual airport, individual aircraft might be assigned a hanger for storage purposes. This relationship can be modeled by first defining an AIRCRAFT class with an attribute called Hanger. Another class HANGER can then be defined that describes hangers. The Hanger attributes of the instances of AIRCRAFT can then contain the object identifiers of HANGER objects. These HANGER objects correspond to the hangers in which the aircraft are housed.

The relationship between aircraft and hangers can also be expressed in the opposite direction. The HANGER class may be defined to have an attribute called Stored_Objects which will contain, as its value, a list of aircraft. In this case, the

[3] Not all object-oriented programming languages support the class concept. For an example of an object-oriented language without classes, see [UNGA87].

Stored_Object attribute of a HANGER instance will contain a list of object identifiers of individual aircraft[4].

Object systems can exhibit considerable flexibility in describing relationships between dissimilar objects. Attribute values may not be restricted with respect to the class of the object. For instance, the Stored_Object attribute can be extended to contain the Object identifiers of different classes of aircraft or even other vehicles.

4.0 INHERITANCE

Complex applications that make use of objects, such as Geographic Information Systems (GIS), often require the definition of many different classes. It is possible to have a separate class definition for each kind of object. However, if some objects are more specific kinds of other objects, it would be natural to take advantage of this relationship when defining classes. This could be done by having class definitions in which a specific class can share or borrow part of the definition of a more general class.

4.1 Superclasses And Subclasses

The mechanism for creating a class definition that derives attributes and methods from another class definition is called inheritance. A class that inherits attributes and methods from another class is referred to as a subclass. The class from which the subclass inherits attributes and methods is its superclass. The concepts of superclass and subclass are analogous to the concepts of generalization and specialization familiar in data modeling methodologies [SMIT77].

In the example, a class called PASSENGER_AIRCRAFT may be defined as a subclass of the class AIRCRAFT. As a subclass, PASSENGER_AIRCRAFT would inherit the attributes and methods of AIRCRAFT. But, PASSENGER_AIRCRAFT may have additional attributes, such as Number Of Passengers and Baggage Weight. Similarly, PASSENGER_AIRCRAFT may have methods that describe operations unique to passenger aircraft, such as procedures simulating loading of passengers and flying holding patterns.

[4]Describing relationships between dissimilar entities is not unique to the object paradigm, but is also an essential characteristic of many other data modeling systems [CHEN76], [CODD81].

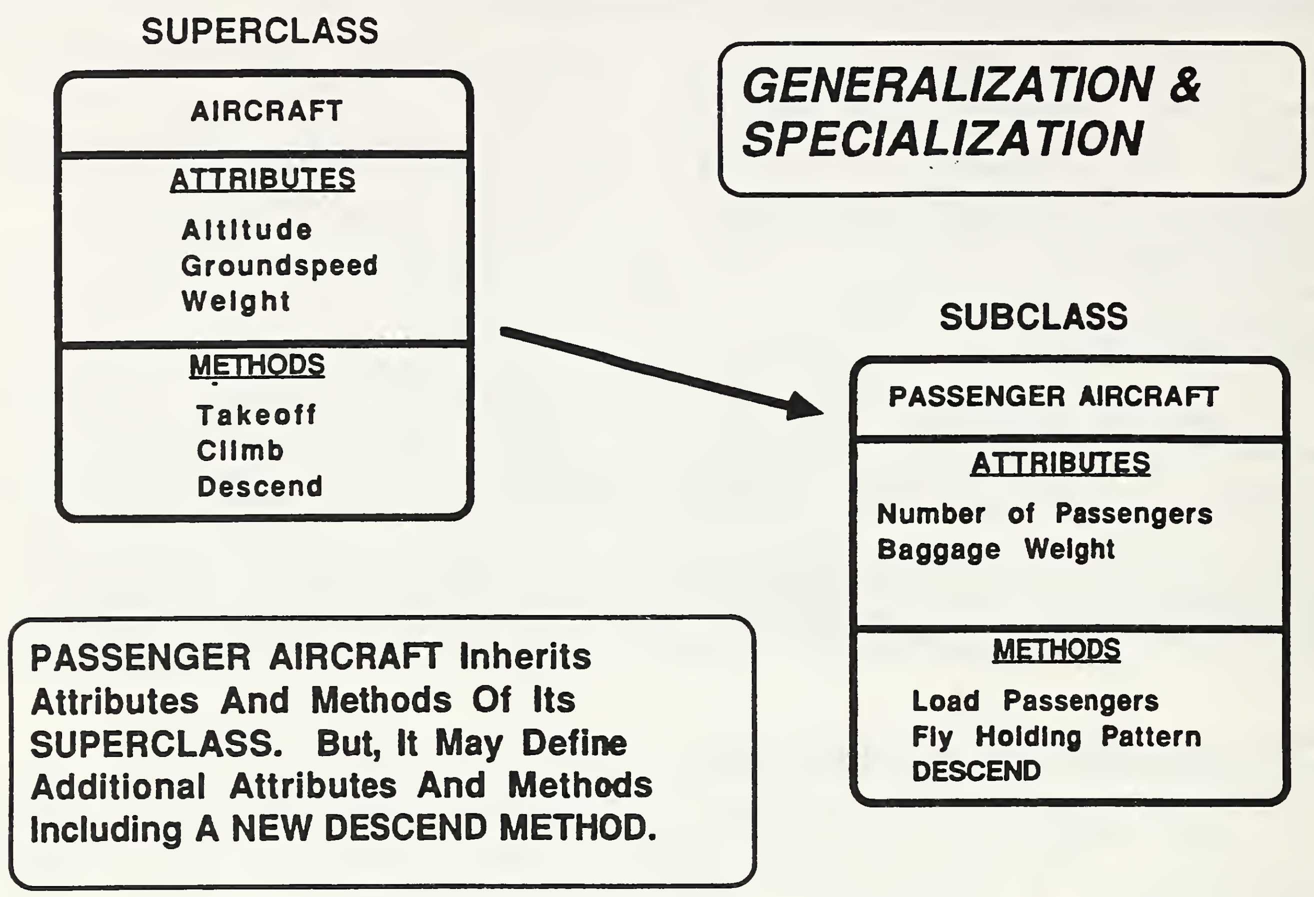


Figure 6. Inheritance In Classes Of Aircraft

In addition, PASSENGER_AIRCRAFT may also define a separate DESCEND method with special constraints on maximum rates of descent. The new DESCEND method defined in PASSENGER_AIRCRAFT replaces, or shadows, the DESCEND method defined for AIRCRAFT.

4.2 Class Hierarchies

Using inheritance, class hierarchies can be created which reflect natural relationships found in the real world. For instance, we may further specialize PASSENGER_AIRCRAFT to create a new subclass AIRLINER.

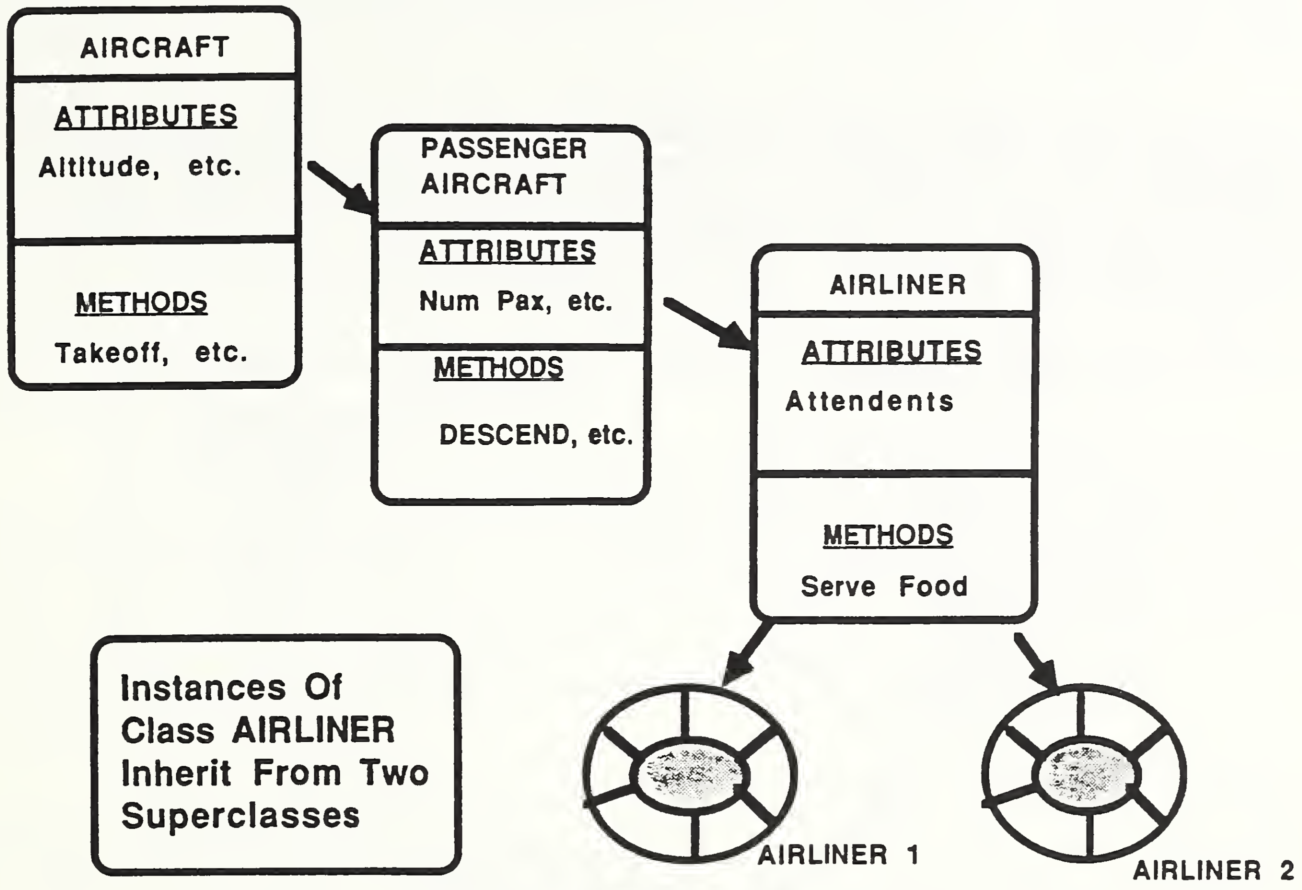


Figure 7. Formation Of Class Hierarchies

Much larger hierarchies can be created that describe complex taxonomies.

Inheritance permits sharing of class definitions. In a computer program, the extensive use of inheritance can substantially reduce the amount of code needed to implement an application. For this reason, object-oriented programs are said to promote <u>code reuse</u>.

4.3 Extensibility

Object systems are inherently extensible. Inheritance allows class hierarchies to be extended by creating new class definitions that reference existing superclasses. For instance, suppose that during the development of the aircraft simulation application, the need arises to model a reconnaissance aircraft. It is easy to define a new class SPY_PLANE and add it to the existing hierarchy by specifying MILITARY_AIRCRAFT as its superclass.

Extensibility has important consequences for software development. Not only can existing applications be easily modified and extended, entire class hierarchies developed in one application can be reused or transplanted to new applications.

4.4 Multiple Inheritance

In many object-oriented languages, it is possible for a class to have more than one superclass. This is known as multiple inheritance. Using multiple inheritance, subclasses can be defined that combine the characteristics of two or more superclasses. As an example, a class SEAPLANE can be defined that inherits the attributes and methods of both AIRCRAFT and WATERCRAFT.

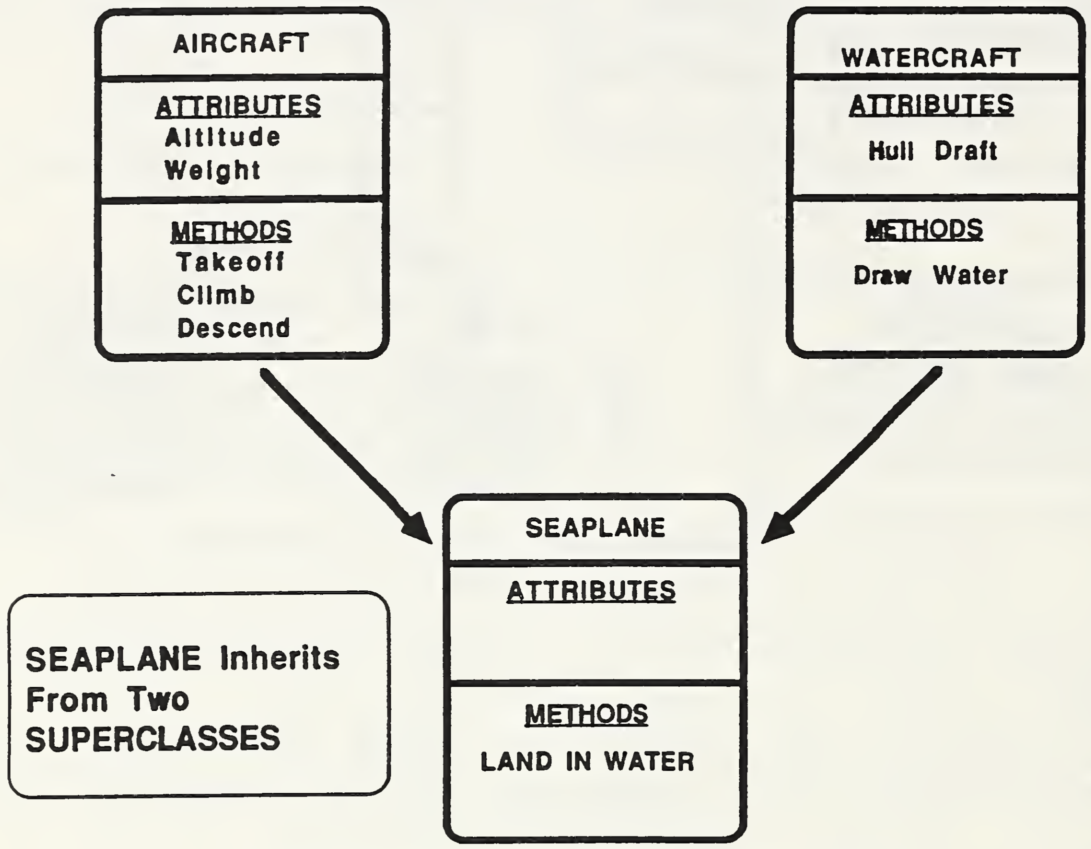


Figure 8. Multiple Inheritance

Multiple inheritance is often advantageous in creating applications with complex modeling requirements, such as geographic information systems. For interested readers, multiple inheritance is discussed in [CANN82] and [CARD90].

5.0 POLYMORPHISM

Polymorphism adds an important dimension to an object's behavior. Polymorphism means that an object's response to a message is determined by the class to which the object belongs.[5] Instances of different classes can be addressed in a uniform manner; e.g., receive the same messages, yet exhibit different behaviors.

In the example, polymorphic behavior can be achieved by first defining two classes of airplanes: PASSENGER_AIRCRAFT and FIGHTER_AIRCRAFT. DESCEND messages can be defined in the protocols of both classes. However, each class defines a different DESCEND method that uses a different maximum rate of descent. For instance, a fighter aircraft can descend much more quickly than a passenger aircraft. When DESCEND messages are sent to instances of either class, a different method responds and performs a different descend operation.

Polymorphism is a powerful aspect of the object paradigm because it permits objects of different classes to be addressed uniformly. The sender of the message does not necessarily have to know the class of the receiving object to successfully accomplish the operation. This can be particularly important in an application with many objects of different classes where it is necessary that each object respond to the same message and perform different behaviors.

Polymorphic behavior is closely associated with <u>run time binding</u>, or <u>dynamic binding</u>. Run time binding means that the selection of the method that responds to the message is made at run time, during the execution of the program. When the message is sent, it is not known beforehand which method will be used. The selection of the method is performed by an internal mechanism maintained by the system for this purpose. In a conventional program, the determination of what code will be executed is made earlier, when the program is compiled.

Polymorphic behavior can be much more complex when inherited methods are invoked. This is particularly so when multiple inheritance is used. The references given at the end of this paper can be consulted for further information.

6.0 COMPOSITE OBJECTS

<u>Composite objects</u>, or <u>complex objects</u>, are among the most recent developments in object technology. A composite object is made up of other objects. In other words, it is constructed from a collection of parts, each of which is itself an object.

[5] More generally, polymorphism refers to being able to apply a generic operation to data of different types. For each type, a different piece of code is defined to execute the operation.

For instance, an AIRCRAFT might be defined as a composite object consisting of separate parts for FUSELAGE, ENGINE, COCKPIT, etc. Each component part is an object and an instance of a class. The process can be extended to produce a <u>part hierarchy</u>, illustrated below in Figure 9.

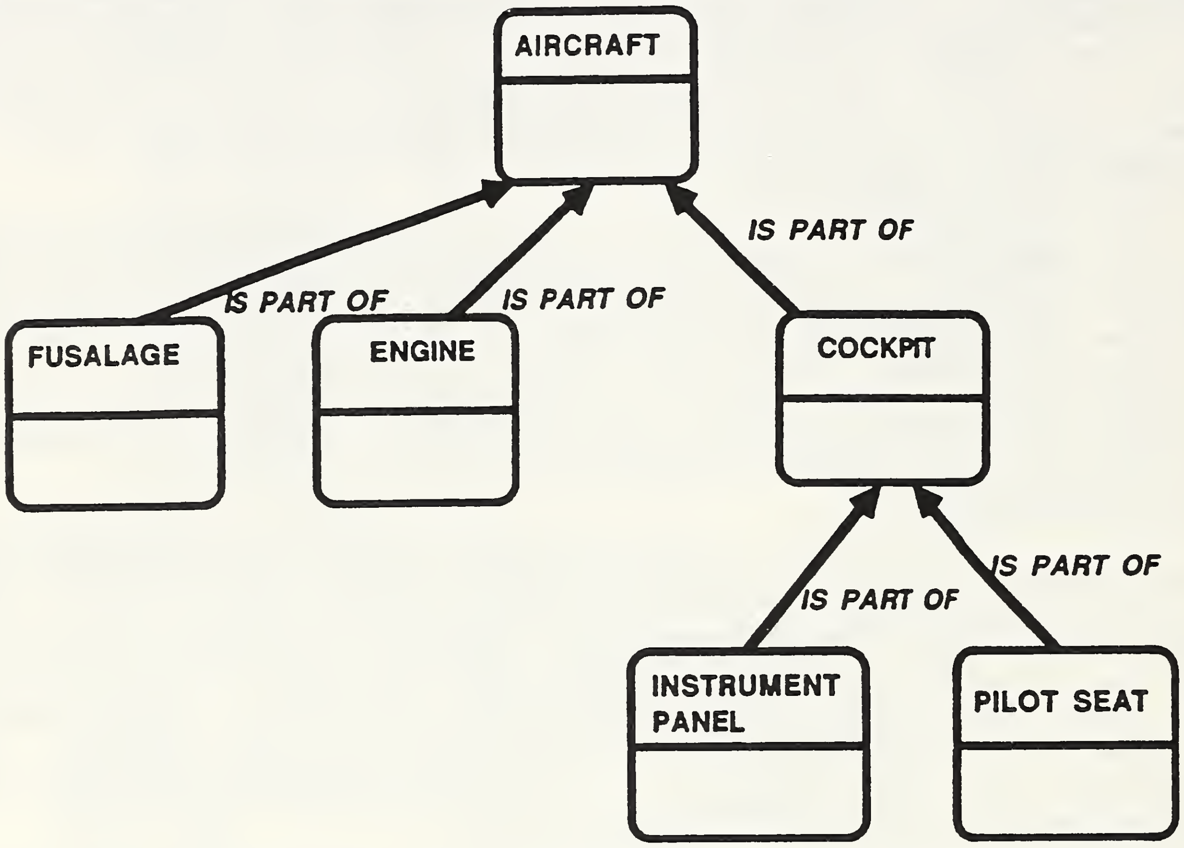


Figure 9. An Example Of A Part Hierarchy

Individual components are said to be in an "Is-Part-Of" relationship to the composite object. For instance, INSTRUMENT_PANEL and PILOT_SEAT are in an "Is-Part-Of" relationship to COCKPIT.

Composite objects are potentially useful in many application domains. Some examples are the modeling of part hierarchies in engineering design systems, and the representing spatial data in geographic information systems.

Composite objects have special requirements for generic operations on their components [DAYA90]. These include operations for manipulation of the composite object as a whole and for manipulation of component subsets. It also includes the ability to define operations which can be propagated to components via relationships. For instance, a message COMPUTE_TOTAL_WEIGHT might be sent to an instance of the composite object AIRCRAFT. The message is propagated by AIRCRAFT to each of its components. If any of these components have subcomponents, the action will be repeated. Each component adds its own weight to that of any subcomponents and returns it to the composite object that it "Is-Part-Of." In this way, the total weight of the entire aircraft is computed.

Experimental implementations which support composite objects are described in [DAYA90], [BLAK87], and [KIMW87].

7.0 CONCLUSIONS AND SUMMARY

This paper has provided a brief overview of the object paradigm, describing its essential features and their application.

Object-oriented programming languages provide many advantages including modularity, extensibility, software reusability, and the capability to easily and clearly model real-world systems. In addition to programming languages, object concepts have been applied with success to graphics systems, software development environments, computer operating systems, and geographic information systems.

The application of object concepts to geographic information systems is a major theme of this publication. Subsequent papers will provide more detail.

8.0 REFERENCES

[BLAK87] Blake, E., and Cook, S. "On Including Part Hierarchies in Object-Oriented Languages with an Implementation in Smalltalk," in ECOOP '87 Conference Proceedings, June 1987.

[BOBR88] Bobrow, D. G. et al. Common Lisp Object System Specification, X3J13 Technical Report 88-002R, June 1988.

[CANN82] Cannon, H. "Flavors, A Non-hierarchical Approach to Object-Oriented Programming," Draft Document 1982.

[CARD90] Cardelli, L. "A Semantics of Multiple Inheritance," in Readings In Object-Oriented Database Systems, Zdonik, S., and Maier, D. (eds.) Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[CHEN76] Chen, P.P.S. "The Entity-Relationship Model - Toward a Unified View of Data," ACM Transactions on Database Systems, 1:1, March 1976, 9-36.

[CODD81] Codd, E.F. "Data Models in Database Management," SIGMOD Record, Vol. 11, No. 2, Feb 1981.

[DAHL66] Dahl, O., and Nygaard, K. "Simula - An Algol-Based Simulation Language," Communications of the ACM, 9, 9, 1966, 671-678.

[DAYA90] Dayal, U. et al. "Simplifying Complex Objects: The PROBE Approach to Modelling and Querying Them," in Readings In Object-Oriented Database Systems, Zdonik, S., and Maier, D. (eds.) Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[FISH89] Fishman, D. et al. "Overview of the Iris DBMS," in Object-Oriented Concepts, Databases, and Applications, edited Kim, W., and Lochovsky, F. ACM Press, New York, 1989, pp. 219-250.

[FORD88] Ford, S. et al. Zeitgeist: Database Support for Object-oriented Programming, Lecture Note 334, 1988.

[GOLD83] Goldberg, A., and Robson, D. Smalltalk-80, The Language and Its Implementation, Addison-Wesley, Reading, MA, 1983.

[HORO83] Horowitz, E. Fundamentals Of Programming Languages., Computer Science Press, Rockville, MD, 1983.

[JOSE89] Joseph, J.V. et al. "Object-Oriented Databases: Design and Implementation," Draft Report from Texas Instruments, 1989.

[KIMW87] Kim, W. et al. "Composite Object Support in an Object-Oriented Database System," in OOPSLA '87 Proceedings, 1987.

[KIMW88] Kim, W. et al. "Integrating an Object-Oriented Programming System with a Database System," in OOPSLA '88 Proceedings, 1988, pp 142-152.

[KIMW89] Kim, W. et al. "Features of the ORION Object-Oriented Database System," in Object-Oriented Concepts, Databases, and Applications, edited Kim, W., and Lochovsky, F. ACM Press, New York, 1989, pp. 250-282.

[LISK77] Liskov, B., Snyder, A., Atkinson, R., and Schaffert, C. "Abstraction Mechanisms in CLU," Communications of the ACM, (20, 8), 1977, 564-576.

[MAIE87] Maier, D., and Stein, J. "Development and Implementation of an Object-Oriented DBMS," in Shriver, B., and Wagner, P. (eds.) Research Directions in Object-Oriented Programming, MIT Press, Cambridge MA 1987, pp 355-392.

[OODB90] Object-Oriented Database Task Group, A Reference Model for an Object Database, Draft Report to X3/SPARC Database System Study Group (DBSSG), Available from Elizabeth Fong, NIST, Technology Bldg. A 266, Gaithersburg MD 20899, 1990.

[SMIT77] Smith, J.M., and Smith, D. "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems, 2(2) June 1977.

[STEF83] Stefik, M. et al. "Knowledge Programming in LOOPS: Report on an Experimental Course," AI Magazine, Vol. 4, No. 3, Fall 1983.

[STRO86] Stroustrup, B. The C++ Programming Language, Addison-Wesley, Reading, MA, 1986.

[UNGA87] Ungar, D., and Smith, R.B. "SELF: The Power of Simplicity," OOPSLA '87 Proceedings, Orlando, FL, 1987, 227-241.

[WEGN87] Wegner, P. "Dimensions of Object-Based Language Design," OOPSLA '87 Proceedings, Oct. 1987, pp 168-181.

Object-Oriented Structures in GIS

Mike Morgan
and
Matt Thorn
Semantic Solutions, Inc.

Barry Glick
Donnelley Geographic Systems

Abstract

Systems employing large amounts of geographic information support a wide variety of applications. Increasingly, these systems support the development of sophisticated spatial arguments in different problem domains. This requirement demands maximum expressiveness in the underlying data model. Object-oriented database (OODB) techniques promise to provide needed support for advanced modeling needs. Specific areas of concern in the development of OODB systems include: 1) Implementation strategies needed to support the complexity of the model; 2) Integration of the data modeling capabilities with programming languages; and 3) Cognitive overhead in support of sophisticated modeling needs. Areas of promise in using the object approach include: 1) Localized, standard models and behaviors support the development of user interface management systems; 2) Direct database support for graphical presentation; and 3) Computational behaviors associated with data elements provide the ability to support analytic procedures. Geographic modeling needs, and the consequent applications, will benefit from the development of semantic models to support multiple levels of access and manipulation.

Introduction

Object-oriented database (OODB) technologies promise to provide added capability to the system developer tool-kit. The advent of new applications, centered around graphical displays, user-oriented models and the integration of databases, are directly supported with OODB. Systems using spatial information, by their very nature, are natural candidates for these advanced modeling abilities. This is especially evident as the user community becomes more sophisticated and applications begin to closely model real world problem areas.

Object-oriented approaches are part of the logical evolution in software tool development. The definitions of Wegner [1] provide a basis of definition for object-oriented approaches. Wegner identifies and discusses three degrees of "objectness" in the object model:

- Object-based - Languages that support the concept of objects. Objects are characterized by models of internal state and behavior. Object behavior is implemented through the use of message passing.

- Class-based - Class-based languages extend the object-based concept by placing every object in a particular class. This mechanism allows the system to provide notions of class modeling, behavior and type derived behavior.

- Object-oriented - The final category defined by Wegner adds the concept of inheritance to the object-based and class-based concepts. Inheritance provides the capability to model generic behaviors with generalization notions while providing an ability to create specific behaviors with aggregation mechanisms.

Wegner's definitions pertain to object-oriented languages. Other recent articles by Kim [2], and Atkinson [3] identify definitions for OODB. A minimalist definition for OODB that we adopt covers two criteria:

1) The system should be a database management system - This criteria means the system must provide all the features found in DBMS's, i.e. persistence and integrity.

2) OODB's must support object-oriented constructs - The modeling capabilities associated with object-oriented languages, such as class and hierarchy/lattice definition and method support, must be included in the database system.

OODB's provide extensions to typical database environments in two critical areas. First, abstract data typing provides modeling beyond flat file types, supporting the development of semantic models of the data sets. Secondly, the association of computational methods with the database supports dynamic behaviors.

Object-Oriented Modeling and Geographic Information

Geographic information is especially suited to the development of OODB models. By their very nature, geographic data sets pertain to real world entities. Applications utilizing this information can take advantage of an ability to model multiple facets of the underlying information base. One possible approach to high-level geographic semantics includes models for:

- Physical entities - The support for physical entities includes both a static and dynamic view of all geographic objects. The static representation captures long term, slowly evolving traits of the entities. Our development efforts, centered around interactive systems, led to the need for a dynamic view of the base objects. Linkage is provided for creation of objects not included in the base model, subsets of the underlying base and thematically derived geographic information. The OODB approach allows all of the objects, static, dynamic or implicitly derived, to be treated in a consistent and uniform fashion.

• Cartographic models - Effective use of geographic information in decision support is augmented by the inclusion of cartographic presentation methods. The OODB approach allows us to develop a process model of cartographic presentation. Overall control issues in base map selection, feature generalization and thematic overlay generation are encoded in a uniform fashion.

• Geometric models - Geometric models are comprised of the operations needed to support metric and topological interactions. The OODB model allows system developers to think in terms of the low-level primitive functions. These in turn can be used to develop the specific, higher-level spatial operations needed by applications.

Following this approach, multiple views of the information sources needed and the operations required are integrated in the underlying information base. The modeling decomposition allows application development to focus on the salient components related to the problem at hand.

Benefits of the OODB Approach

In our experience, there are many benefits to be gained from the OODB approach. Some of these benefits are:

1) Modularity in data design - Encapsulation, both in terms of state and behaviors, allows designers to focus on the generic aspects of the data base entities. The focus on objects at this level leads to the development of standard sets of vocabularies and interactions enhancing the reusability of the information base.

2) Integration of higher level semantics - Multiple granularities of data are maintained. Support for low-level data entities are

extended with high level constructs. Vocabularies can be designed to support the application level, building on the lower levels.

3) Support for interactive mapping environments - Explicit maintenance of geographic entities and their corresponding cartographic presentation provides direct support for the development of the displays needed for interactive decision support. Direct manipulation interfaces for the maps and mapped objects are supported, leaving the application developers free to develop appropriate models of the interfaced process.

Selected References

[1] Dimensions of Object-Based Language Design, Peter Wegner, OOPSLA 1987

[2] Object-Oriented Databases: Definition and Research Directions, Won Kim, IEEE Trans. on Knowledge and Data Engineering, Sept. 1990

[3] The Object-Oriented Database Manifesto, Atkinson, et. al. 1988

[4] Knowledge Base Structures for Object-Oriented GIS, M. Morgan and B. Glick, GIS/LIS 1988

Object-oriented Database Approaches in GIS

Mike Morgan

NIST Conference

August 23-24, 1990

Briefing Overview

1. Preliminaries

2. Modeling Issues

3. One Approach

4. Implementation Issues

5. Summary/Lessons

Preliminaries

Defining Object-oriented (language)

Object-based

- Supports objects as a feature
- State, operations, message passing
- Ada, Actor

Class-based

- Every object is in a class
- Type (class) derived behavior
- Clu

Object-oriented

- objects + classes + inheritance
- hierarchical or lattice structures
- method resolution
- CLOS, Smalltalk

Defining Object-oriented (database)

Criteria 1: Should be a DBMS

- Persistence, secondary storage management, concurrency, recovery and ad-hoc query

Criteria 2: Should support object-oriented constructs

- Complex objects, object identity, encapsulation, classes (types), inheritance, over-riding and late binding, extensibility and computational completness

Benefits of Providing OODB Models

- Supports interactive, aided and automated analysis.
- Supports generation of appropriate, application specific information displays.
- Allows for flexible query of multiple information types.
- Supports integration of heterogeneous sources thru easy linkage at the data level.

Costs/Dangers in OODB

- Modeling overhead - Unconstrained expressiveness can lead to trouble.
- Implementing the "engine" - Database implementation is no longer a simple flat file.
- Accessability requirements - Multiple access needs need lead to multiple indexing schemes.
- Problem areas for OODB - What are reasonable domains for the added expense.

Modeling Issues

Data Modeling Approaches

- Hierarchical Model - Collection of data records linked as a hierarchy, i.e. one way links.
- Network Model - Multi-linked extension to hierarchical model with links possibly having meaning.
- Relational Model - Tabular representation with rows representing relationships between values.
- Entity-Relationship Model - Conceptual model of data entities with explicitly defined relationships.
- Semantic Data Models - Attempt to model (explicitly) higher-level meanings of data entities.

Static Modeling Support

- Encapsulation - Defining data entities (objects) in terms of internal state and behavior.

- Object identity - Objects are unique entities, not a normal form.

- Composite objects - Multiple object descriptions treated as a "whole" (creation, deletion ...)

- Inheritance - Sharing of behaviors increases potential for re-use. Scheme used (i.e. multiple or single) has impact on expressiveness.

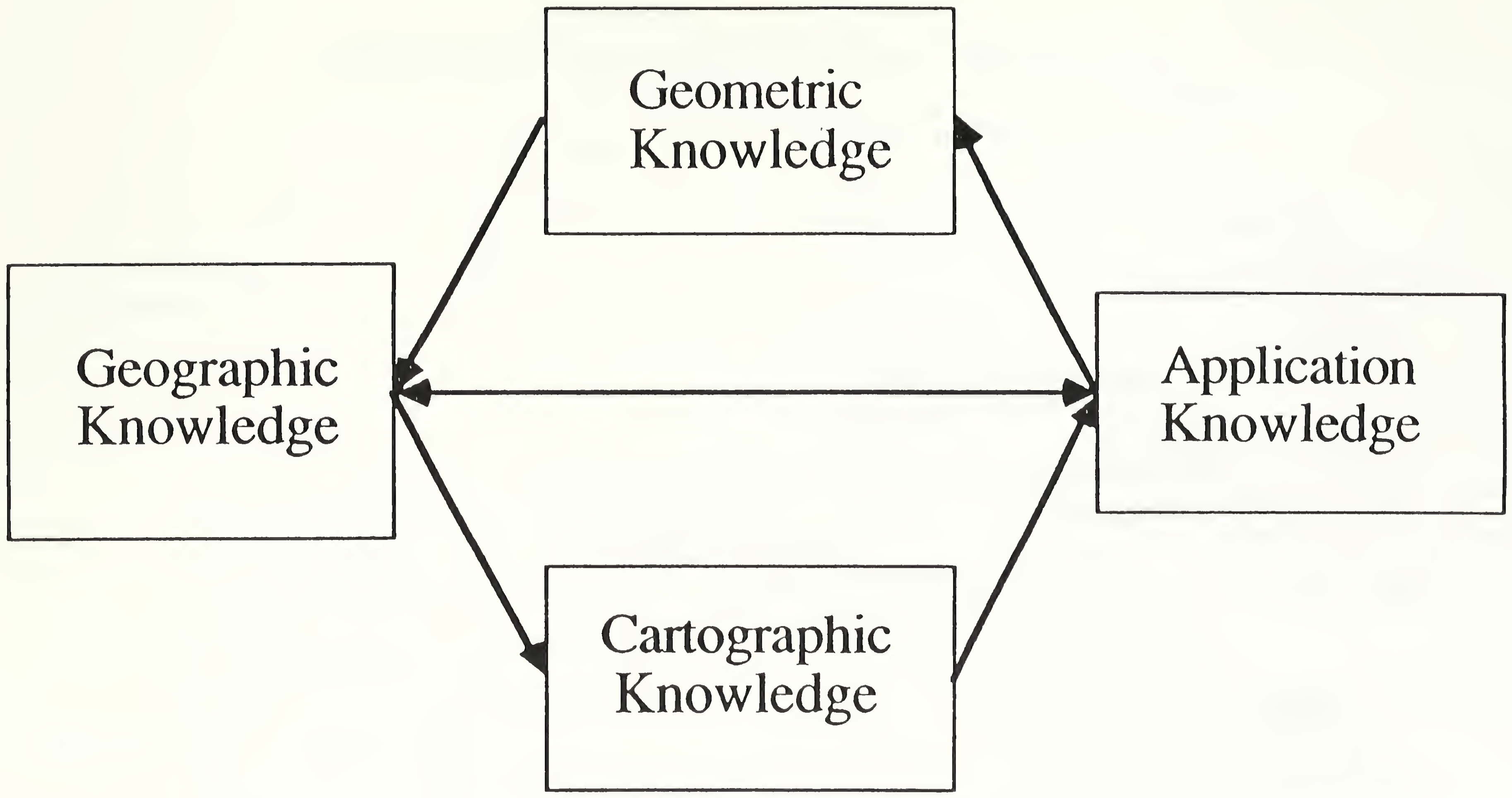
Dynamic Modeling Support

- Computational behavior - The process of interacting with the database for a value can involve method invocation.

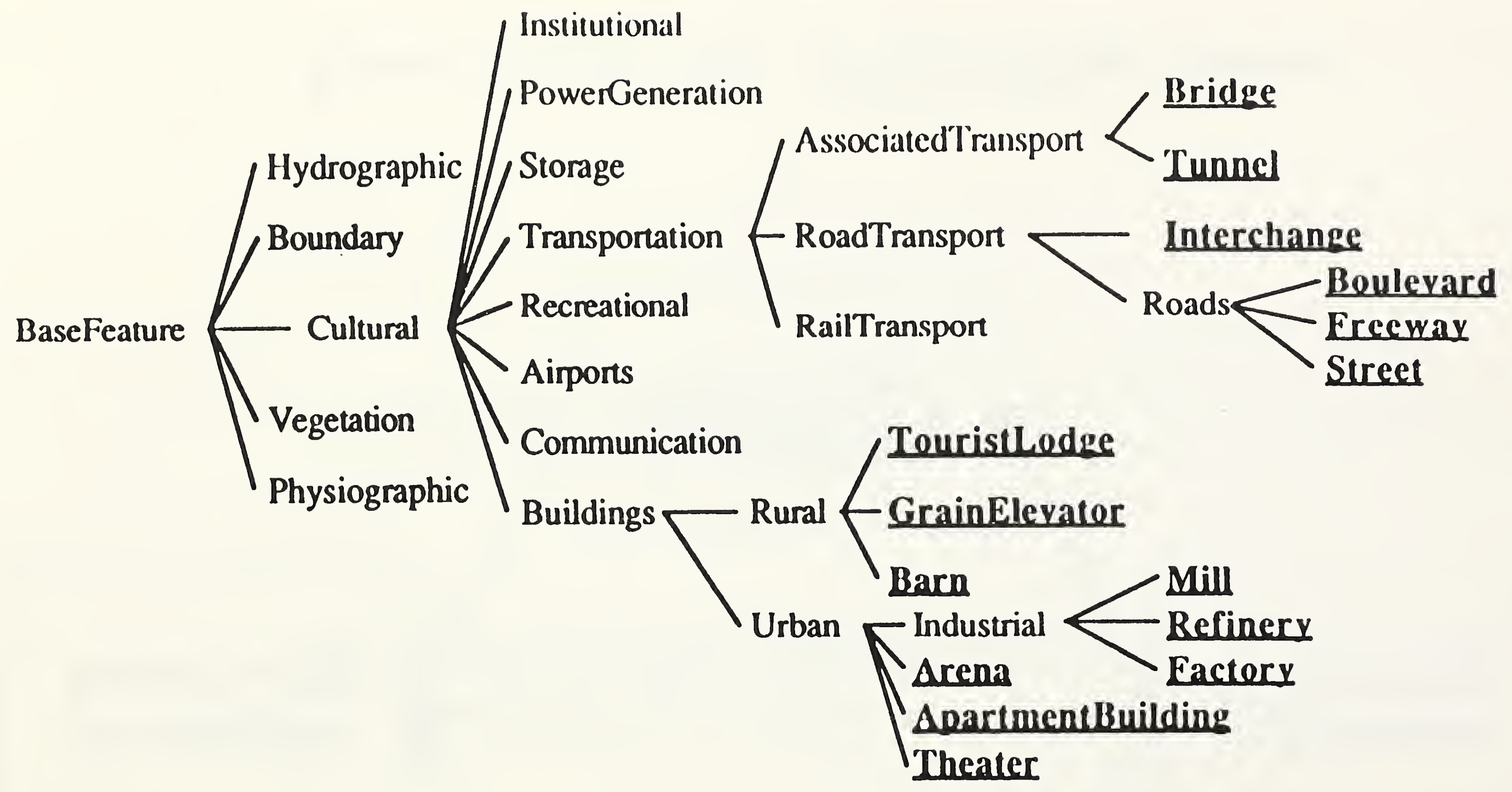
- Multi-methods - Method invocation is really dispatching on a variable argument list.

- Polymorphism - Binding environments for the method invocation should be allowed to be determined at run time.

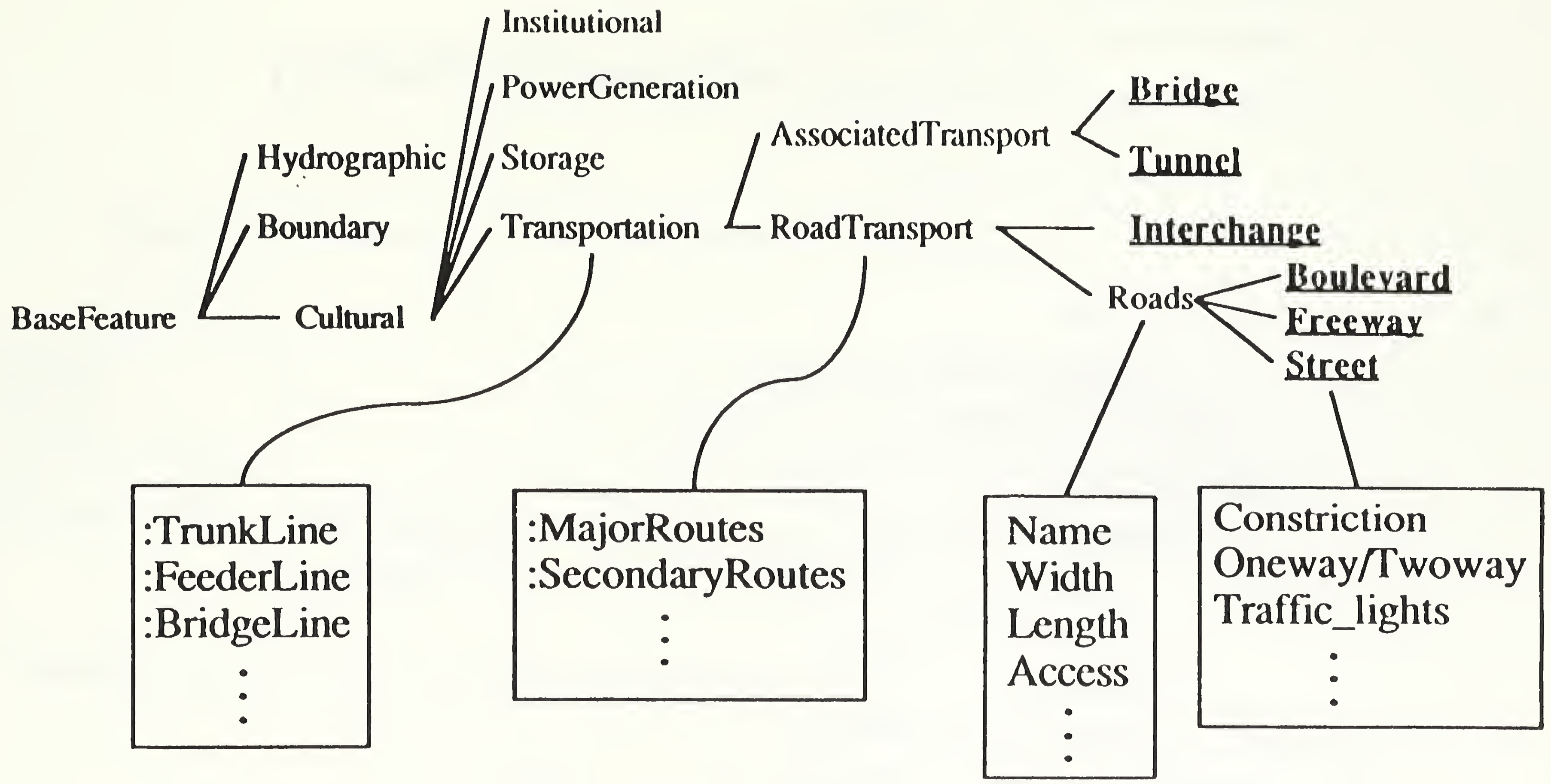
One Approach to Knowledge Structuring



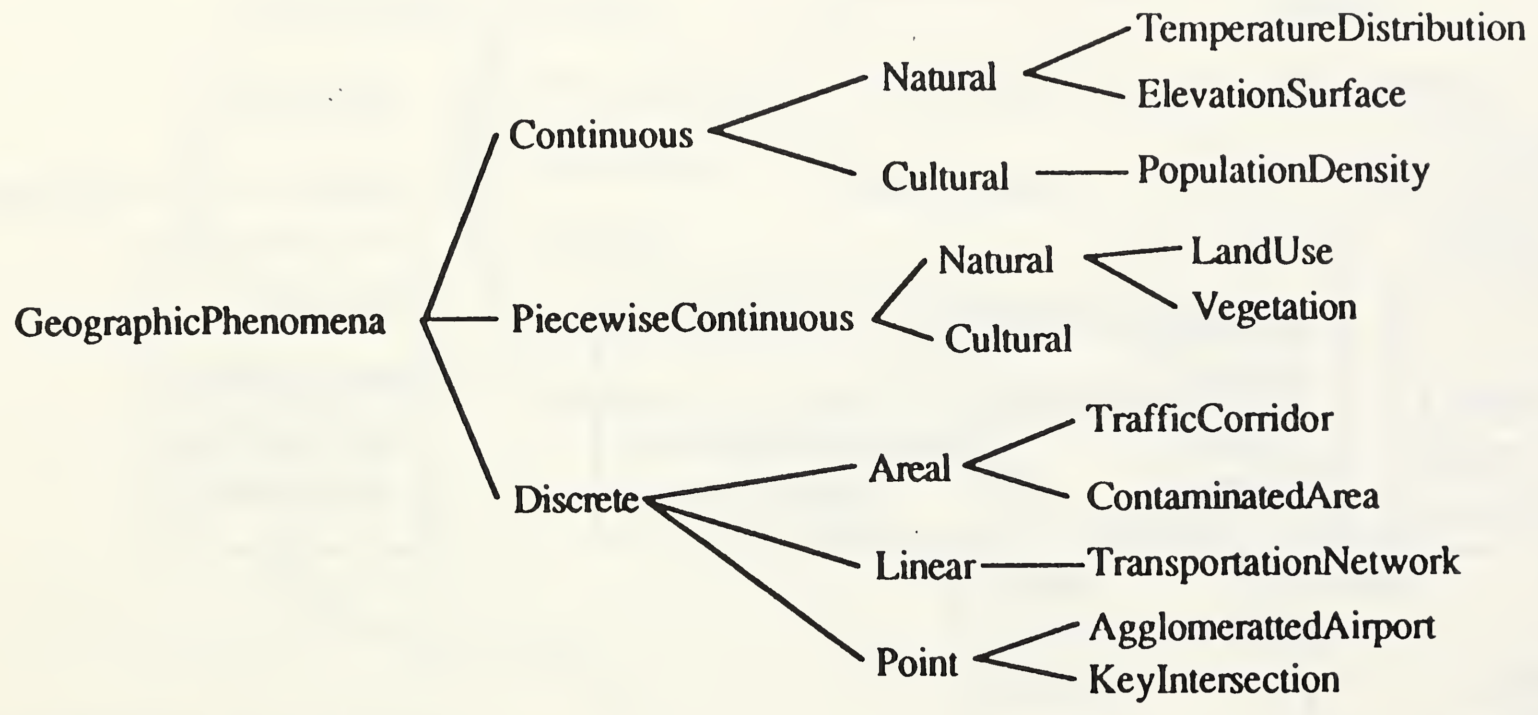
System Level View of Knowledge Sources



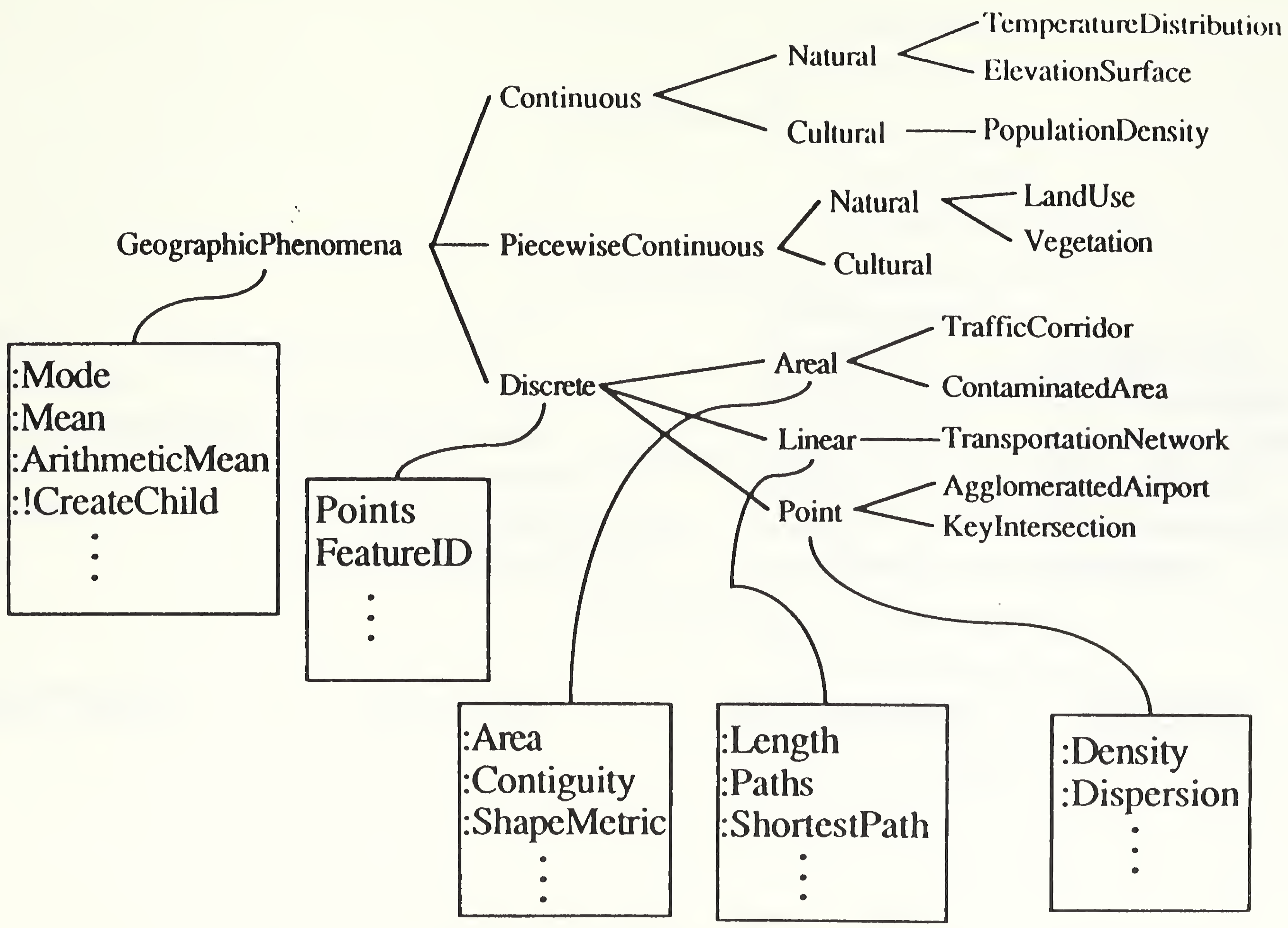
Portion of the BaseFeature Hierarchy



Annotated Portion of the BaseFeature Hierarchy



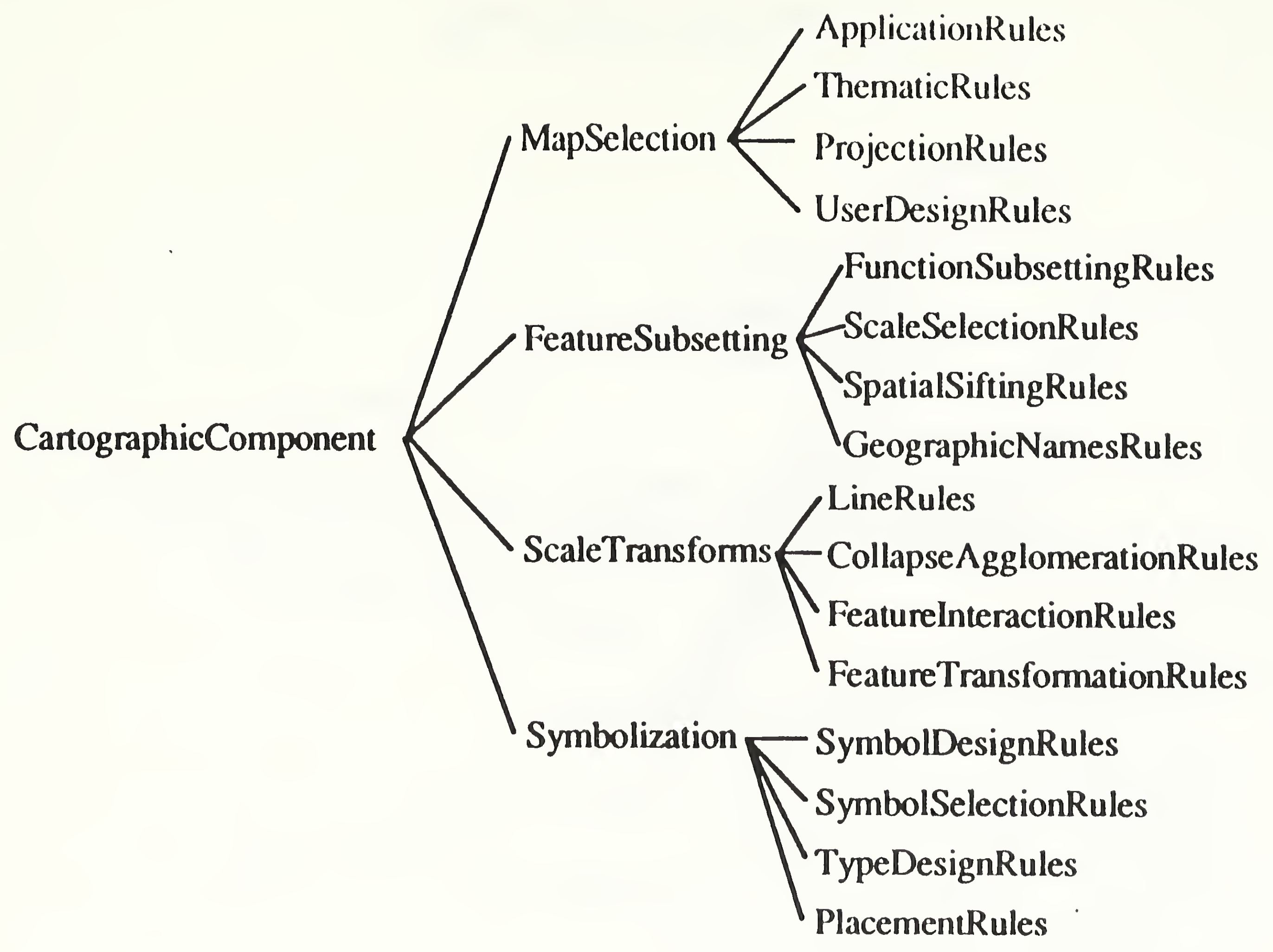
GeographicPhenomena Hierarchy



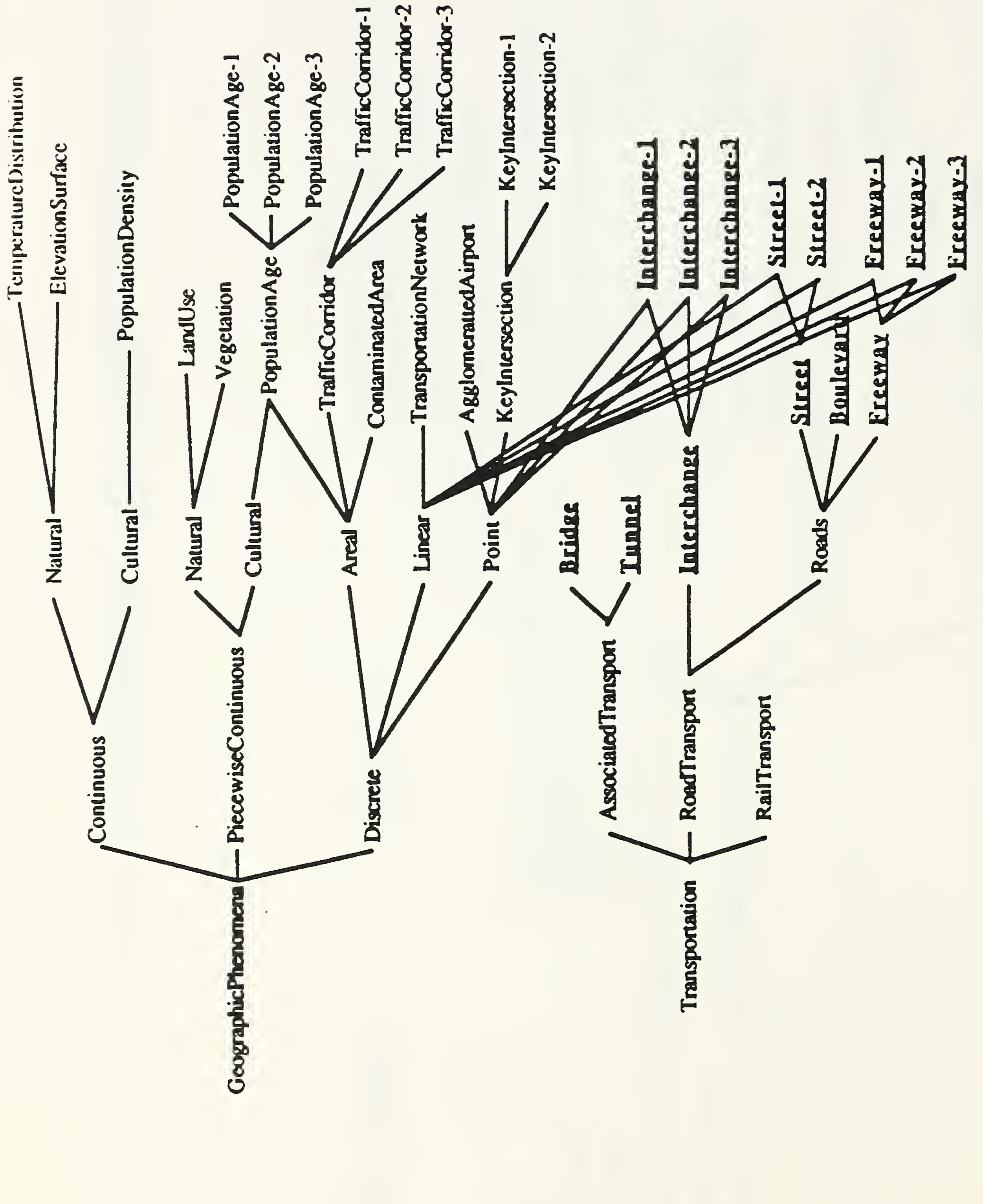
Annotated GeographicPhenomena Hierarchy

Types of Operators and Queries with Examples

DB Query	Derived	Geometric	External
Mill	PrimaryTrafficCorridor	LocationOf	ToxicSpill
Freeway	SecondaryTrafficCorridor	NextTo	EntertainmentEvent
Bridge	ContaminatedArea	WithIn	SportingEvent
ApartmentBuilding	KeyIntersection	EntranceTo	PropagationModel
Interchange		Isolated	EconomicModel
Tunnel		ParallelTo	WeatherFront
		DistanceFrom	



Cartographic Knowledge Hierarchy



Example Hierarchy

Implementation Issues

Implementation Strategies

- DBMS v. Language Extension - How tightly coupled should the implementation be to a given language.

- Physical data model - Does the physical model represent the object (un-normalized) view or is it in some more conventional form (i.e. relational)

- Clustering/Indexing - Sophisticated environments can benefit from user control of the index mechanisms.

Implementation Strategies (cont.)

- Inheritance conflict resolution strategies - Depending on the type and binding strategy, several alternatives are possible.

- Concurrency - Database locking and buffer management strategies must be developed to support mutiple users. General issues of data integrity and security must be addressed.

Summary/Lessons

Summary

- Object-oriented design and implementation supports modularity and extensibility; A knowledge-directed modeling capability.
- OODB design models higher level geographic concepts while maintaining lower level geographic reference.
- OODB supports the creation of MMI designs for flexible, direct manipulation of objects in a knowledge directed fashion.
- The required environments exist now for implementing these techniques.

Lessons Learned

General:

- OO-type systems become very knowledge/data intensive
- It's a long term effort to develop the "tid-bits" of knowledge required in a complete system.

Knowledge Sources:

- Effective system modeling requires both static and dynamic view of information.
- Support for geographic reasoning will require close ties to analytic operations.

Lessons Learned (cont.)

Knowledge Representation:

- Multiple hierarchies for multiple views.
- Unfortunately life is not hierarchical

Knowledge Access:

- Levels of abstraction support analysis and presentation.
- Vocabularies are independent of applications.
- Vocabularies are dependent on applications.
- Objectness helps focus vocabulary development

General Principles

Principle 1 - In data and knowledge intensive systems semantic modeling mechanisms are imperitive.

Principle 2 - Efficient system modeling requires multiple views on knowledge sources.

Principle 3 - Knowledge must be accessable at all levels.

Principle 4 - Effective presentation supports accessability.

Principle 5 - Take advantage of all the knowledge in the system all the time.

Geographic Application Principles

Principle 1 - Strive for compatibility with expected data sets.

Principle 2 - Don't limit yourself to form and content of existing geographic data sets.

Principle 3 - Understand user requirements to support vocabulary development.

Principle 4 - Understand user requirements for topology.

Principle 5 - The key to future geographic processing technologies will be the integration and understanding of analysis.

Principle 6 - Cartography may be art but you still need it.

Selected References

Dimensions of Object-based Language Design,
Peter Wegner, OOPSLA '87

Object-oriented Databases: Definition and Research Direction,
Won Kim, IEE Trans on Knowledge and Data Eng. Sept '90

The Object-oriented Database Manifesto
Atkinson, et. al. '88

Knowledge Base Structures for Object-oriented GIS
M. Morgan and B. Glick , GIS/LIS '88

GIS/Object Technology Demonstration

Matt Thorn, Mike Morgan

Semantic Solutions, Inc.
8950 Villa La Jolla Drive, Suite 2200
La Jolla, California 92037

Presented at the NIST GIS Standards Laboratory Technology Integration Workshop, Gaithersburg, Maryland, August 23-24, 1990.

Introduction

This paper is a description of the Geographic Information System/Object Object Technology Demonstration presented at the GIS Standards Laboratory's Technology Integration Workshop. The demonstration is an application of the geographic knowledge structuring concepts presented in workshop's talk entitled, "Object-Oriented Database Approaches in GIS." The demonstration is an adjunct to work performed for the National Ocean Service's (NOS) Nautical Charting Research and Development Laboratory (NCRDL).

The demonstration depicts a computer mapping program developed to illustrate the utility of maintaining multiple, object-oriented views of geographic space. The demonstration focuses on a general spatial data model and its use in supporting the automatic generation of nautical charts. Additionally, the program exemplifies data model support for interactive applications and integration of related geographic information. The program also serves as a test-bed for experimenting with spatial data management in an object-oriented environment.

The program was developed on a Sun Workstation using the X Window System and C++. It uses a prototype 1:40,000 - 1:80,000 scale nautical feature data set obtained from NCRDL. The area of coverage is a 1° x 1° block at the entrance to the Houston Ship Channel in Galveston, Texas. The data set includes an extensive set of water and ground related features including channel segments, buoys, lights, piers, wrecks, shorelines, roads, and railroads. In total, there are approximately 28,000 features.

Knowledge Representation

Geographic Knowledge Representation

The demonstration employs a novel (Morgan and Glick, 1988) representation for storage and organization of geographic data. The data are organized into two representations: static and dynamic. The static representation maintains basic feature data. The dynamic representation maintains more complex geographic information that is process oriented. Both representations are briefly introduced below.

The Base Feature Hierarchy

The static representation of the geographic data, known as BaseFeature, is in some respects similar to typical feature data sets, as it consists of elemental features. However, BaseFeature features are not organized into a flat representation, but into a hierarchy. The leaves of the hierarchy correspond to the canonical data set. The leaves' parents represent successive levels of geographic abstraction. For example, HardSurfaceRoad has Road as a parent, Road has Transportation as a parent, and so on until the hierarchy's root

AllFeatures. A class of features may have multiple parents, so BaseFeature is actually a lattice, i.e. River has both Transportation and Hydrography as parents.

BaseFeature serves two purposes: organization and knowledge abstraction. An important characteristic of the organization is that it allows the same data to be viewed in different ways. Consider the queries "select all transportation features with characteristic X" and "select all hydrographic features with characteristic Y." Both queries may result in sets of features that include rivers, but the perception of rivers is different in both cases. The structure, via inheritance, supports factoring and reuse of common attributes and functionality by allowing knowledge to be encoded at appropriate lattice levels. This knowledge abstraction assists applications by permitting them to work at levels of abstraction appropriate to their tasks.

The Geographic Phenomena Hierarchy

Geographic applications often require complex, process oriented, geographic information. This information is often dynamic in nature and is generally not found in BaseFeature. The structure that captures this information is known as GeographicPhenomena. Like BaseFeature, GeographicPhenomena is organized as an inheritance lattice. The geographic information it contains derives from three general classes: discrete, piecewise, and continuous phenomena. The classes are then broken down as necessary. In the nautical domain, one can imagine continuous classes like TideModel, Bathymetry, and Weather. The class TideModel, for example, might provide methods for deriving tide surges. The methods' results could be instances of a piecewise class TideSurge. More appropriate to the nautical chart generation example, GeographicPhenomena contains the discrete and piecewise continuous classes ShipChannel, ContinuousShoreline, Waterfront, and Harbor. Instances (features) of these classes are linked to features in BaseFeature. The features contained in GeographicPhenomena are more complex than those of BaseFeature. BaseFeature contains channel segments, buoys, and lights, which compose ship channels, but it does not contain ShipChannels which are particular combinations of these features. Similarly, BaseFeature does not capture the notions ofWaterfront and ContinuousShoreline that are required (along with ShipChannel) to define the feature Harbor. Often complex features in GeographicPhenomena are composed of features from both GeographicPhenomena and BaseFeature.

The advantages of GeographicPhenomena are similar to those of BaseFeature; it is organizational and knowledge is strategically factored. More important, however, is that by capturing and encapsulating complex geographic concepts and their operations, the underlying data representation is elevated to a level approaching the application domain. This simplifies processing of the data since it is possible to discuss, reason, and use the concepts and features as entities.

Cartographic Knowledge Representation

Cartographic knowledge, the information necessary to determine how to display geographic information, is also viewed, at least for the purposes of this discussion, as a hierarchy. In reality, the knowledge representation may take several forms: it may be declarative, as in an expert system; procedural, as in features' operators; or a combination of both. For the purpose this of demonstration, the cartographic information is considerably simplified. It consists of a theme or essence of a nautical chart, feature selection rules, and feature presentation rules.

The Demonstration

The spatial data model briefly presented above is inherently object-oriented. It was natural to use an object-oriented language to implement the demonstration as well as the object-oriented data model to store the data. Other traditional data models, i.e. relational, network, etc., do not afford the necessary expressiveness to effectively implement the data representation. The implementation language is C++.

The demonstration consists of three parts: a basic mapping capability, nautical chart generation examples, and a data fusion example. Underlying this is a basic, persistent object manager that manages access to the geographic data and a C++ class that implements the map. Feature access is accomplished using quad trees. There are approximately 56 thousand small (~400 - ~2k bytes) objects stored in the database.

Basic Softcopy Support

The basic mapping capability allows viewing of BaseFeature elements by permitting toggling of their classes. All data access and transformations are performed at runtime. Support for panning and zooming of the map within a range of scales (1:1M - 1:10K) is also provided. Feature attribute information may be obtained by selecting map features with the mouse. The mapping capabilities, while elementary, demonstrate interaction with the data representation in a softcopy environment.

Nautical Chart Generation

The nautical chart generation examples are the primary focus of the demonstration because they illustrate interactions with and the utility of the spatial data representation. For simplicity, the cartographic process is limited to considering only simple feature generalization as it applies to scale change. This also includes notions of feature selection and symbolization. Future work will address more complex cartographic processes.

The nautical chart demonstration consists of displaying a range of charts starting at a scale of 1:250,000 and proceeding to 1:100,000, 1:50,000, 1:25,000, and 1:15,000. Each chart consists of all the features in the previous charts plus additional features. As the scale changes from one chart to the next, the displayed representations of the features change to suit the chart's scale and theme. Displayed features include those from both BaseFeature and GeographicPhenomena. Those from GeographicPhenomena are most interesting because they represent complex features cartographers grapple with, such as ShipChannels, ContinuousShorelines, Airports, Waterfronts, and Harbors.

When a chart is generated, a *display* message is sent to the object that describes the chart (this object specifies a chart's theme and scale). Subsequently, appropriate categories of features are selected, sent *display* messages, and, if appropriate, displayed. Each GeographicPhenomena feature determines its own representation based on chart scale and theme. The cartographic knowledge is encapsulated within the features. If display of a feature is inappropriate, perhaps because the feature is more suited to a larger scale chart, the feature is not displayed. For example, only major ShipChannels are displayed at the 1:250,000 scale, and they are represented on the chart by only their centerlines, even though they consist of channel segments, buoys, and lights. At the 1:25,000 scale, however, all ShipChannels are displayed, and they are represented by symbolized representations of their channel segments, buoys, and lights. Imagine the difficulty of automating the display of ship channels if solely their parts existed without any organizing structure like ShipChannel.

Extensibility and Data Fusion

Critical to remember is that while the demonstration focuses primarily on chart generation, the data and the data model in particular are not limited to supporting this single application. Charting is simply one perspective on the data. There are other applications that would use the same data in completely different ways. For example, consider an application supporting harbor management. It would use the same data, but for different purposes: buoy maintenance, channel dredging, etc. It would also require additional related data to be integrated with the charting data. An additonal benefit of the object-oriented nature of the spatial data model is that it simplifies the process of fusing collateral geographic data sets with the existing data. The demonstration provides an illustrative example of this by displaying a thematic representation of ship to pier capacities using some canned United States of America Army Corps of Engineer data, which contains information on the number of ships particular piers serve, fused with the NOS pier data.

Conclusion

The demonstration illustrates the powerful utility of an object-oriented spatial data model that supports multiple views of geographic information. The data model consists of a static and a dynamic structure. The static structure BaseFeature captures basic geographic features and their abstrations. The dynamic structure GeographicPhenomena captures complex, process oriented, geographic information which does not exist in BaseFeature. The demonstration provides several examples of the application of the data model. The primary emphasis of the demonstration is how the representation can be used to support automatic generation of nautical charts. Additionally, examples of softcopy interactions with the data representation and data fusion are shown.

References

Morgan, M. and Glick, B. 1988. Knowledge base structures for object-oriented GIS. Paper presented at GIS/LIS '88, San Antonio, TX.

NOAA'S EXPERIENCE WITH OBJECT PROGRAMMING/DATA BASES AND EXPERT SYSTEMS

Dave Pendleton , Artificial Intelligence Group
Nautical Charting R&D Laboratory
Nautical Chart Division/Coast and Geodetic Survey
National Oceanic and Atmospheric Administration
Rockville, Maryland 20852

Introduction

The National Oceanic and Atmospheric Administration's (NOAA) Nautical Charting Research and Development Laboratory (NCRDL) has used object-oriented programming/data bases since 1987 to explore the application of artificial intelligence programming techniques, specifically expert systems, to automated cartography. The NCRDL conducts research in cartography, hydrography, and photogrammetry to support the Charting and Geodetic Service (C&GS) national navigational chart production program.

The NCRDL, within the C&GS, is a component of the National Ocean Service (NOS), the civilian agency which produces nautical charts for navigation in U.S. coastal and estuarine waters, the Great Lakes, and the Coastal Zone. It also produces the Nation's aeronautical navigational charts, a wide range of geodetic data and services, and maintains the national network of geodetic control monuments which defines the basic geographic framework for all national mapping and charting activities.

Like many Federal and State agencies over the past 15 years, NOS has been implementing automated charting and Geographic Information Systems (GIS) with the goal of interactively producing multiple types of charts and related products from a single scale- and product-independent data base of chart feature data as depicted in Figure 1. The current major effort is to implement a major chart production system using traditional tools and techniques, and then augment the compilation workstation tools with expert system features and capabilities

This paper presents an overview of our efforts to adapt object-oriented and expert systems techniques to the problems of automated cartography as perceived by the NCRDL. The approach is intended to be at a tutorial level with an emphasis upon concepts, rather than details. A secondary goal is to present the material so that the discussion can apply to the interests of a wider audience than digital cartographers.

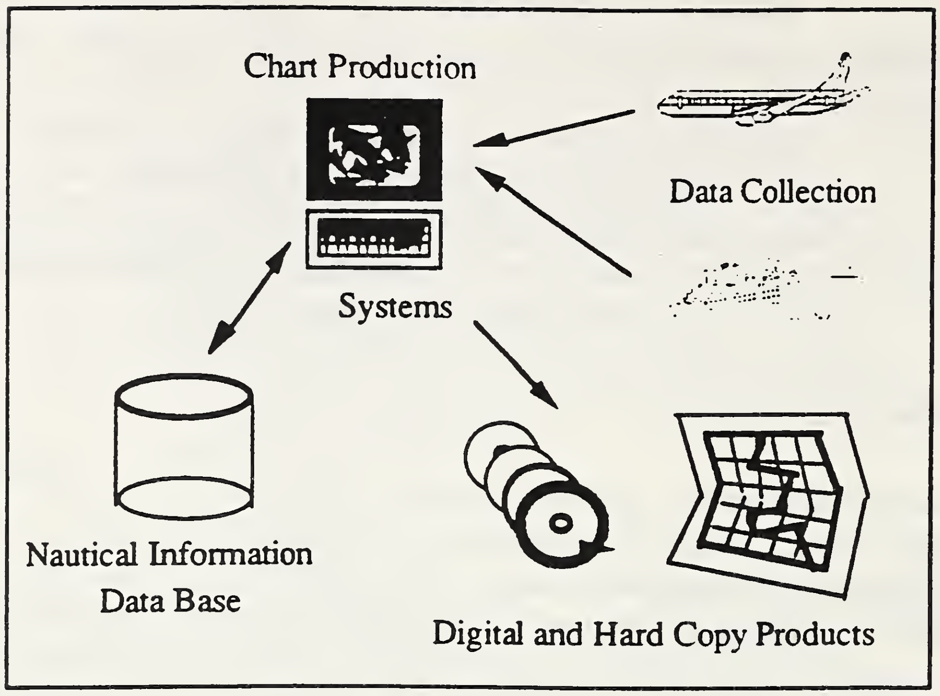


Chart Production
Data Collection
Systems
Nautical Information Data Base
Digital and Hard Copy Products

Figure 1. NOS Chart Production

Like the history of the classical problem of automatic language translation, significant automation of mapping and charting operations has proven to be elusive and deceptively difficult. NOS has implemented advanced data collection systems in the air and on the seas, but creation of the Nautical Information Data Base and the Chart Production System to manipulate it graphically and textually continues to be a long and difficult process.

Background

The NCRDL is turning to expert systems, object-oriented programming, object-oriented data models, and related emerging technologies because the use of traditional automation techniques has not been entirely successful in meeting modern requirements of digital cartography and GIS. These complex requirements, such as feature generalization when performing automatic scale change, have proven to be limiting factors in achieving the economic benefits promised by automated cartography.

NOAA is not alone in this regard. In fact, the entire Federal mapping and charting community, including: the U.S. Geological Survey, the Defense Mapping Agency, the Census Bureau, Environmental Protection Agency, Federal Emergency Management Agency, Soil Conservation Service, Bureau of Land Management, the Forest Service, and hundreds more federal, state, and local agencies are also trying to overcome these same fundamental and quite difficult technical problems on the way to modernizing their cartographic and GIS production systems.

To understand the reasons for the approach taken in NOAA's applications, to be discussed both in this and other papers in these proceedings, one should first have a basic understanding and appreciation of the fundamental technical difficulties of automating chart compilation. Aside from the major problem of creating and initializing the large cartographic data bases involved, the most important technical difficulties fall within one or more of the following categories:

(1) the **automated placement of feature names** and other text labels without obscuring other labels or adjacent feature symbology;

(2) **map generalization at change of scale** involving feature collapse and coalescence, with emphasis on primary feature characteristics using alternate symbolization, if necessary;

(3) the **detection and resolution of conflicts** between symbols and/or text competing for the same physical area of the map; and

(4) **decluttering operations** on masses of data in a given area in order to include features from overlapping areas at different scales.

These operations are further complicated by difficulties inherent in the unorthordox nature of the chart feature data itself. Spatial data of this type is composed of both graphical and nongraphical components, whereby a feature, such as a road, has both a graphic depiction, involving the icons used to symbolize it, together with associated attribute values, such as its name, its material composition, its usage as a primary or secondary road, and other "traditional" data fields.

The kinds of data manipulations required of such a system are very involved. Fundamental to each, however, is the requirement to combine this complex spatial data, which was collected by a variety of techniques at different scales and resolutions, and match or combine it with data of entirely different types within possibly overlapping and/or intersecting spatial areas and have the resulting graphic depictions be correct. In this case "correct " means, in addition to the usual desire for correct numerical attribute values, that the displayed compilation places emphasis on those features requiring emphasis and suppresses or eliminates others while maintaining the esthetic effects achieved by traditional cartographic methods. Of course, the compiled depiction can change entirely as the scale for the same geographic area changes.

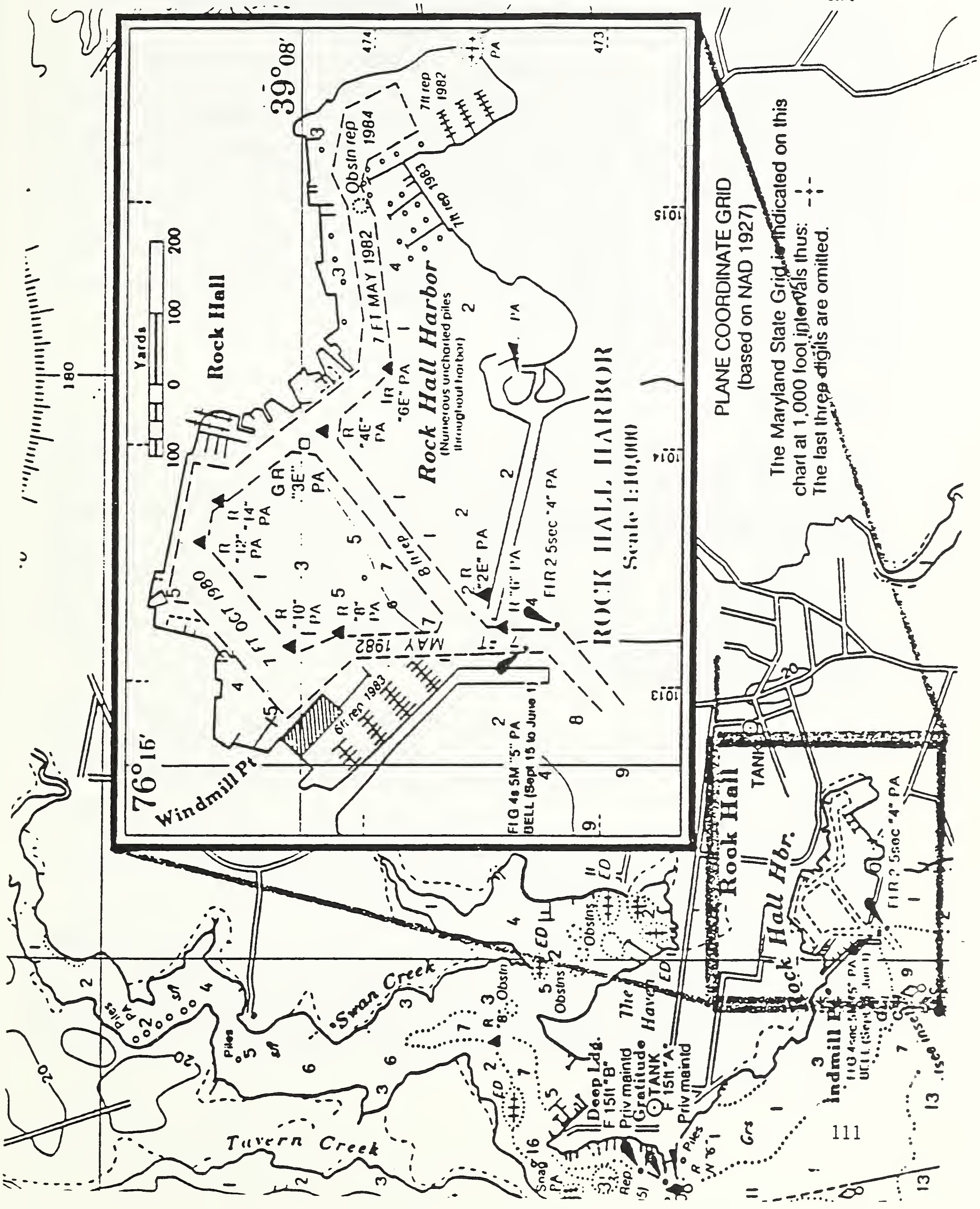
To illustrate the difficulties listed above , the section of a NOAA nautical chart shown as Figure 2 is a typical example of the cartographic depiction problem. This chart contains an inset or portion of the main chart enlarged to a more detailed scale, in this case 1 to 10,000 from 1 to 25,000. Without going into detail as to the exact meaning of all the symbols shown, the dashed lines are ship channels, the sounding numbers denote water depth in feet, and one can see piers, pilings, ship wrecks, and buoys.

The point to notice is that the geographical area on the main chart does not show the same level of detail as that shown in the inset, even though the data is present in the data base. What is not so apparent are the additional features, also in the data base, that are **not** shown, even at the 1:10,000 scale. If all available sounding data were displayed, for example, the entire water area would appear as a single black mass, obscuring all other features as well as the individual sounding values themselves. The text for labels is carefully placed so as to not impact the feature icons while, at the same time, positioned so as to intuitively indicate what is being labeled (note the curved "Windmill Pt").

The main chart is a **generalization** of the area of the inset with an approximation made to the shape of the water line and, except for the channels, most other features in the data base are not shown at all at that scale. In order to construct the inset, the system must change the scale and know which features show up at the new scale, their type and size of icons, their geographic locations, which features to omit in case they conflict (overlay or intersect) with more important icons, and how to place the text such that no features or other text is obstructed.

To be useful in a high-volume production system, the generalization operation, for example, must be performed on a network of graphics workstations from a single scale and chart independent feature data base. Because of the possibility of chart overlaps in common geographical areas, it is unacceptable to allow the system to maintain and access a separate data base for each chart to be produced. This is one of the fundamental constraints placed upon the design of modern digital cartographic systems.

Figure 2. Example Portion of A NOAA Nautical Chart



It is important to realize that these chart depictions, called "compilations," are maintained accurate to scale so that distance measurements can be taken directly from the chart by the mariner. Accordingly, precise geographic positions and the spatial relationships of features for a given area and scale must be maintained. Thus, in constructing the depiction, lateral movement of the icons for critical features, such as ship wrecks, rocks, pilings, and other potential hazards to safe navigation, is not allowed. In some cases, e.g. ship wrecks in close proximity, icons may be rotated in relationship to one another in order to resolve the conflict at a given scale, but they cannot be otherwise moved. As a last resort, alternate icons can be selected or the entire compilation can be depicted in a different way to convey the same information to the mariner.

Objectives

The NCRDL's Artificial Intelligence Group was formed to explore, develop prototypes, and evaluate solutions to these difficult technical problems using emerging automation techniques such as expert systems, object-oriented programming languages, object data bases, GIS, and related technologies. The long-range objective is to provide software tools that can be embedded into the latest version of the agency's ANCS II (Automated Nautical Charting System, Version 2), which will be the new system used to ascession, maintain, and manipuate figure 1's Nautical Information Data Base [3].

By embedding expert systems into ANCS II, we hope to break through conventional technology's technical barriers and apply these emerging techniques to enhance the system's capabilities to automate specific chart compilation functions. This concept is depicted in figure 3, in which a knowledge base of highly structured expert system rules is accessed by a set of expert system tools being driven by the processing of a conventional data base of nautical chart feature information.

The expert system tools consist largely of: (1) a rule interpreter module, commonly called an inference engine in expert systems parlance, (2) a data base of rules called an knowledge base, and (3) a graphical user interface consisting of windows, pop-up menus, etc., all accessable via a mouse and keyboard.

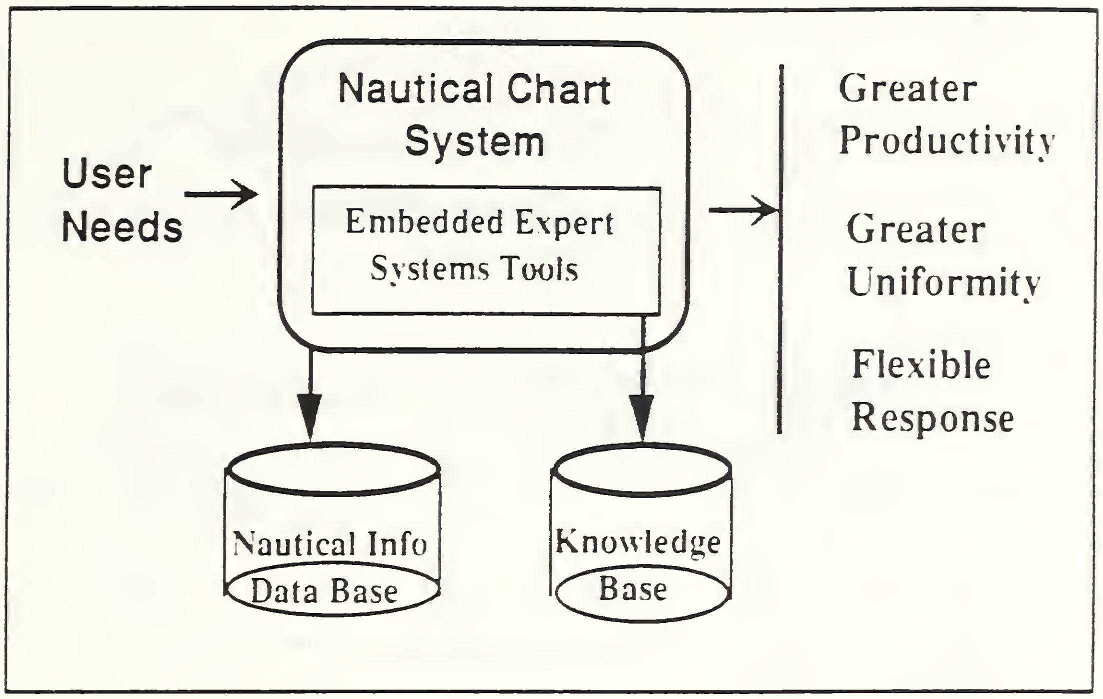
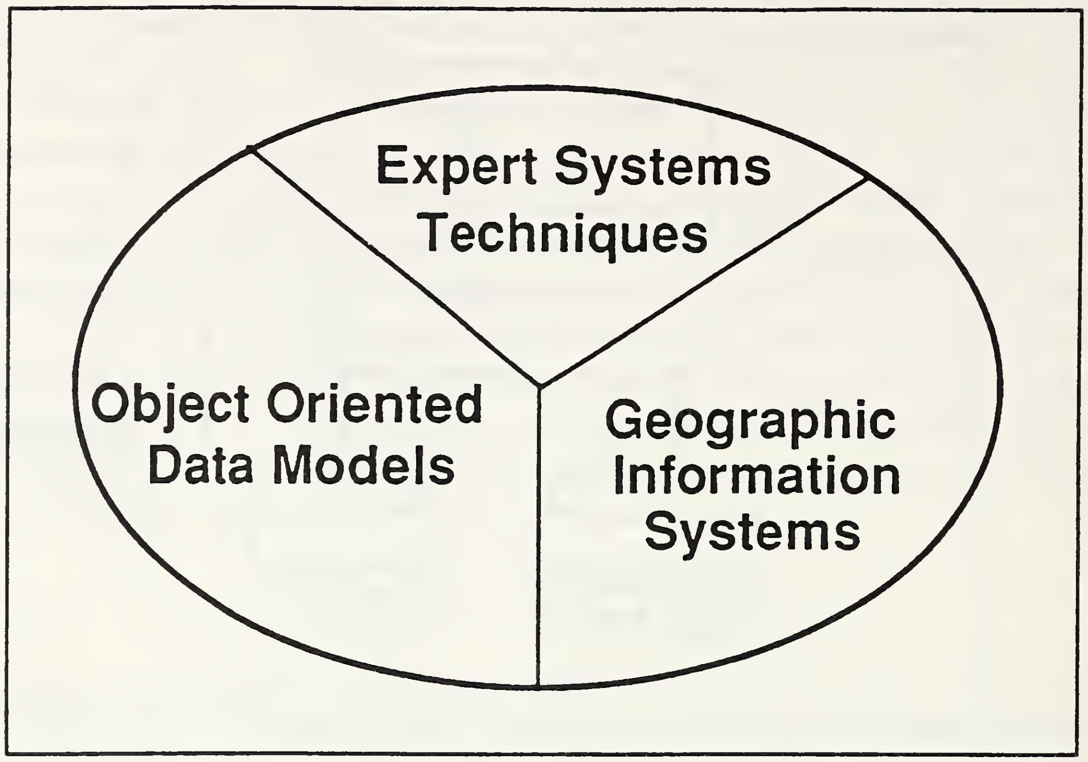


Figure 3. Concept of Enhanced Charting System

Technical Approach

It was clear from the beginning that the traditional approach to system development using the specification-design-code-debug sequence would not be appropriate for the knowledge-based system. The approach selected was to develop and evaluate separate prototypes of each major kind of chart depiction problem, as previously discussed above, and then integrate them under a unified design once the details were known.

The development of each prototype was sequenced through the major phases of problem definition, enabling technology evaluation and selection, test-bed facility enhancement, software development, and iterative refinement through all these steps until a working prototype resulted. This flexible approach has proven quite successful and resulted in a series of working prototype knowledge-based systems. A major outcome of this open design approach was the identification and application of three distinct kinds of enabling techniques and their integration under a unified design. This concept is depicted in figure 4.



Expert Systems Techniques
Object Oriented Data Models
Geographic Information Systems

Figure 4. Integrating Three Leading-Edge Technologies

Previously, artificial intelligence/expert systems, object-oriented programming/data models, and geographic information systems were new and distinct fields. Not only are they distinct but, like the field of automated cartography, each is still in the process of emergence from the laboratory and gaining maturity in the real world. This pioneering work at NOAA has demonstrated that such techniques can be integrated to provide a powerful new paradigm for solving highly complex spatial data processing problems that are very difficult to approach in other ways.

The Prototypes

Table 1 summarizes the results of this effort in terms of the prototypes developed at the NCRDL for specific chart compilation problems.

Prototype	Problem Area	Primary Result
1986 Shipwreck	Proof-of-Concept	Evaluation-Application Chart Compilation Model.
1987 Object Data Model	Knowledge & Data Representation, Spatial Data Mgt	Spatial Object Data Model; Object DBMS Experience.
1988 Interactive Expert Editing System	Text Placement Conflict Detection and Resolution	Point Feature Labeling; Constraint-based Search and Inference.
1988 CartoAdvisor	Knowledge Base	Knowledge Base Tool; Expert Sysem Shell.
1989 KB-GIS Phase I	Scale Change, Generalization, Spatial Data Mgt	GIS Capabilities; Generalization; Feature Collapse; Feature Coalescence; Feature Symbolization.
1990 Carto-Associate	Integrated Object-Oriented Design	Composite Objects; Expert System/GIS; Spatial Object Data Model; Conflict Detection and Resolution; Scale Change and Generalization; Text Placement.

Table 1. Evolution Of NOAA's Cartographic Prototypes.

The Shipwreck Prototype was the first proof-of-concept demonstration which explored the potential of expert systems techniques to problems of chart compilation. It was written in the logic programming language Prolog and, for feasibility, focused upon symbol selection and limited conflict resolution for a single NOAA chart feature, the ship wreck [1].

The Object Data Model prototyped NOAA charts and their features as objects with behavior. This work was programmed in Smalltalk-80 object programming language (Xerox, Inc.) and used the GemStone (Servio Logic, Inc.) object data base management system (refer to the paper by Williams in these proceedings) [5,7].

The Interactive Expert Editing System explored the problems of automated text placement on NOAA's charts. The problem was limited to placing text for point features appearing on NOAA nautical charts, such as buoys. It was written in the C systems programming language and used the InforMap III (Synercom, Inc.) GIS [2].

The CartoAdvisor Prototype was an extension of the shipwreck prototype which was completely rewritten using the LISP-based expert system shell Personal Consultant Plus (Texas Instruments, Inc.). The Shipwreck's small knowledge base was extended and powerful graphics capabilities were added. It has evolved into a tool for creating and debugging knowledge bases for individual chart features [1,4].

The KB-GIS (Knowledge-Based Geographical Information System) Phase I Prototype applied object-oriented GIS-type operations to scale change and generalization problems and proved the benefits of exploiting multiple views of geographic space and higher-level geographic entity/GIS concepts. It was written in the C++ object-oriented language and used the X-Window System (MIT), the InterViews object-oriented user interface (Stanford University), and an object-oriented Mapping/GIS shell (Semantic Solutions, Inc.). A Phase II KB-GIS project to extend and complete this work is planned for 1991-1992 [8].

The CartoAssociate Prototype began as an extension of the object data model prototype and evolved into a platform on which the results of all previous prototypes will be combined under a coherent, integrated design. It is written in Smalltalk-80 and uses the Analyst user interface (Xerox,Inc.), the Humble expert system shell (Xerox, Inc.), as well as the GemStone object DBMS (Servio Logic, Inc.). The CartoAssociate will serve as an automated chart compilation test bed for exploring advanced solutions to the problems of conflict detection and resolution, feature evaluation and symbolization, scale change and generalization, and text placement. It will become the foundation for developing, testing, and migrating a family of expert system/object-oriented/GIS tools onto C&GS's new automated nautical chart production system as it comes on-line [2,3,4,5,6,7,8].

The Experimental Cartographic Facility

The series of prototypes discussed above were developed on NCRDL's hardware/software test-bed platform called the Experimental Cartographic Facility (ECF). This is a networked configuration of engineering workstations, personal computers, and peripherals, such as plotters and printers, that evolved along with the series of prototypes discussed above. The layout is as shown in figure 5 below, with the DEC VaxStationII/GPS serving as the network host and data/object server.

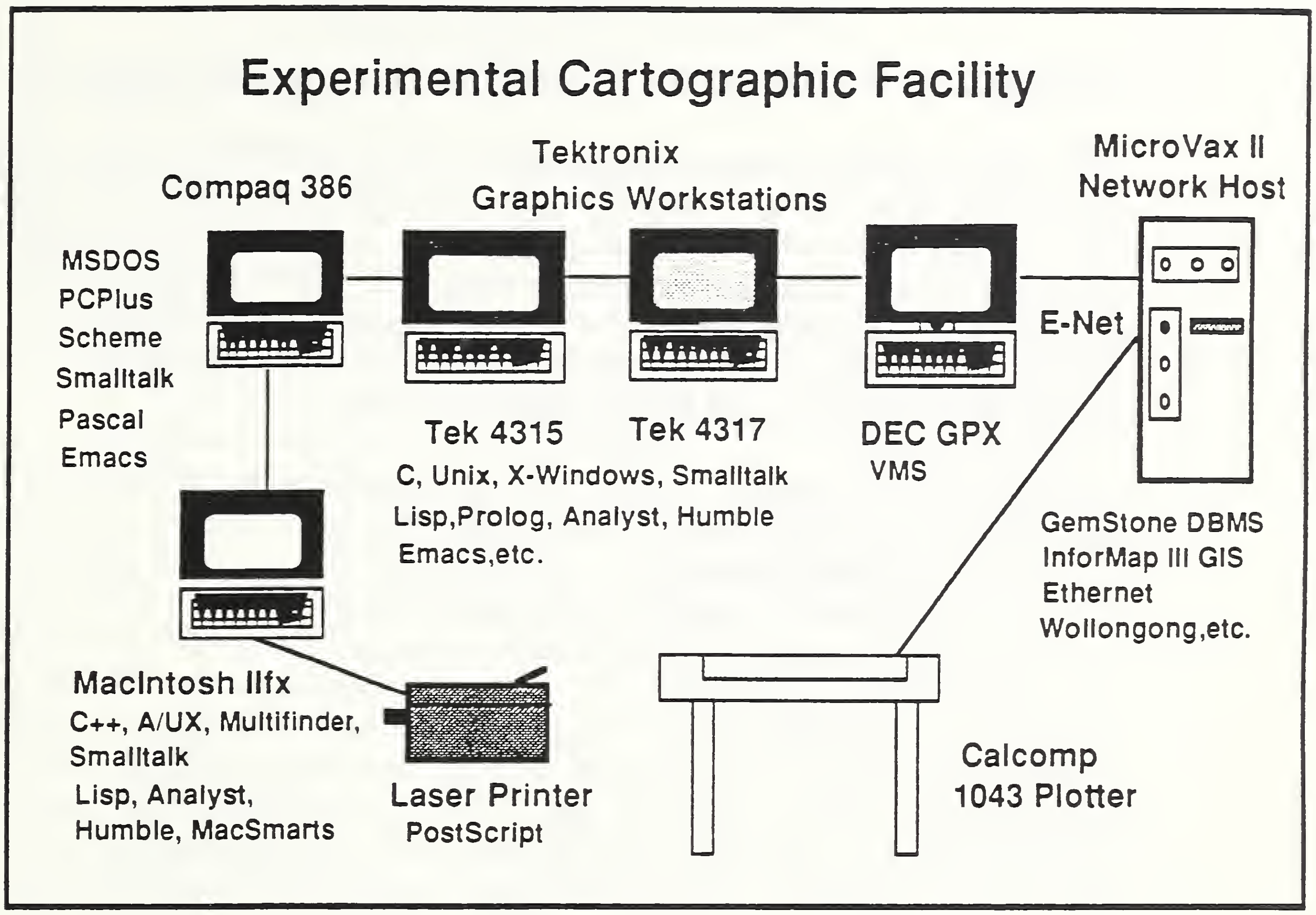


Figure 5. Hardware/Software Test-Bed For Object-Oriented Cartographic Expert Systems Prototype Development.

As the ECF evolved, it became a unique open architecture facility for developing cartographic/GIS prototypes involving knowledge-bases, expert systems, and object-oriented programming/data management. It has a full complement of artificial intelligence languages, several expert systems shells, the leading object-oriented languages, and an advanced object-oriented data base management system. The workstations are UNIX based with 19 inch high-resolution color screens. The MacIntosh and Compaq personal computers also are equipped with 19 inch color screens. The network host will be upgraded to a Sun SparcStation 2 in the near future.

Object-Oriented Programming And Expert Systems Concepts

The work-in-progress demonstrations given at the NIST Workshop reflects the present status of our current projects, the CartoAssociate and the KB-GIS. However, in order to gain a perspective on the spatial application of the powerful programming techniques used in their development, it is first necessary to have some understanding and appreciation of the concepts of modeling with object classes, class inheritance, and the expert systems ideas of rule-based inference and model-based reasoning.

Modeling With Classes. Almost every problem-solving approach can be guided and enhanced by the use of some kind of abstract model. A model is a simplified view of the structure and primary characteristics of the real-word entities comprising a problem to be solved. The use of a model permits de-emphasis of secondary issues and retains a focus upon the critical elements. It also provides context information for making valid inferences of secondary facts when they are needed, without having to retain and maintain volumes of details about the entities and their relationships. With origins in the field of simulation, object-oriented languages, such as Smalltalk-80, and object data base management systems have built-in facilities for constructing, manipulating, and interrogating complex models of the entities comprising a problem.

Figure 6 illustrates how the key characteristics of entities and their relationships can be captured as *class and subclass definitions* in an object-oriented language. Moreover, because these classes are defined as data types in the object program, they become templates for *instances* (records) and their *instance variables* (data fields). They also define the valid operations that may be performed on the problem's data types through the class and instance *methods* (program code) they encapsulate. In Smalltalk, for example, these user-defined data types are complete extensions to the programming language and are just as valid as the built-in types such as "integer," "floating point," and "character."

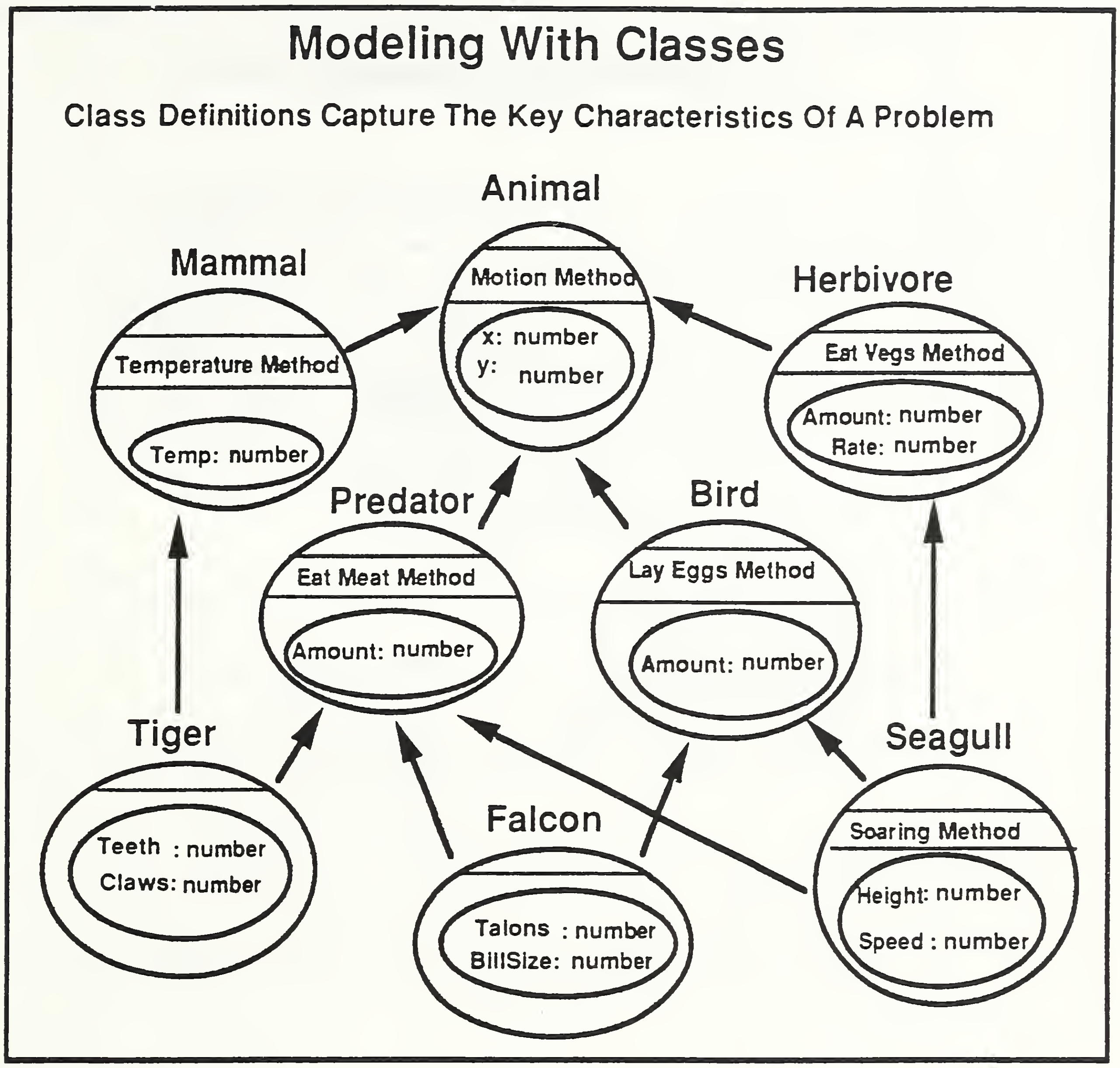


Figure 6. Capturing The Structure Of Entities With Classes

The key concept to realize about figure 6 is that the subclasses of parent classes, e.g., "Animal ->Bird ->Seagull," denotes one *superclass-subclass hierarchy* chain, where the subclasses include only those characteristics *different* from the parent class. All other subclass characteristics are *inherited* from the parent class and are known throughout the programming system as such. This facility makes it easy to add refinements to the model in order to construct as detailed and complex a model as required for the problem. This is further illustrated in figure 7 in which a new data type called WhiteTiger has been defined. WhiteTiger automatically inherits all the characteristics (i.e., methods and instance

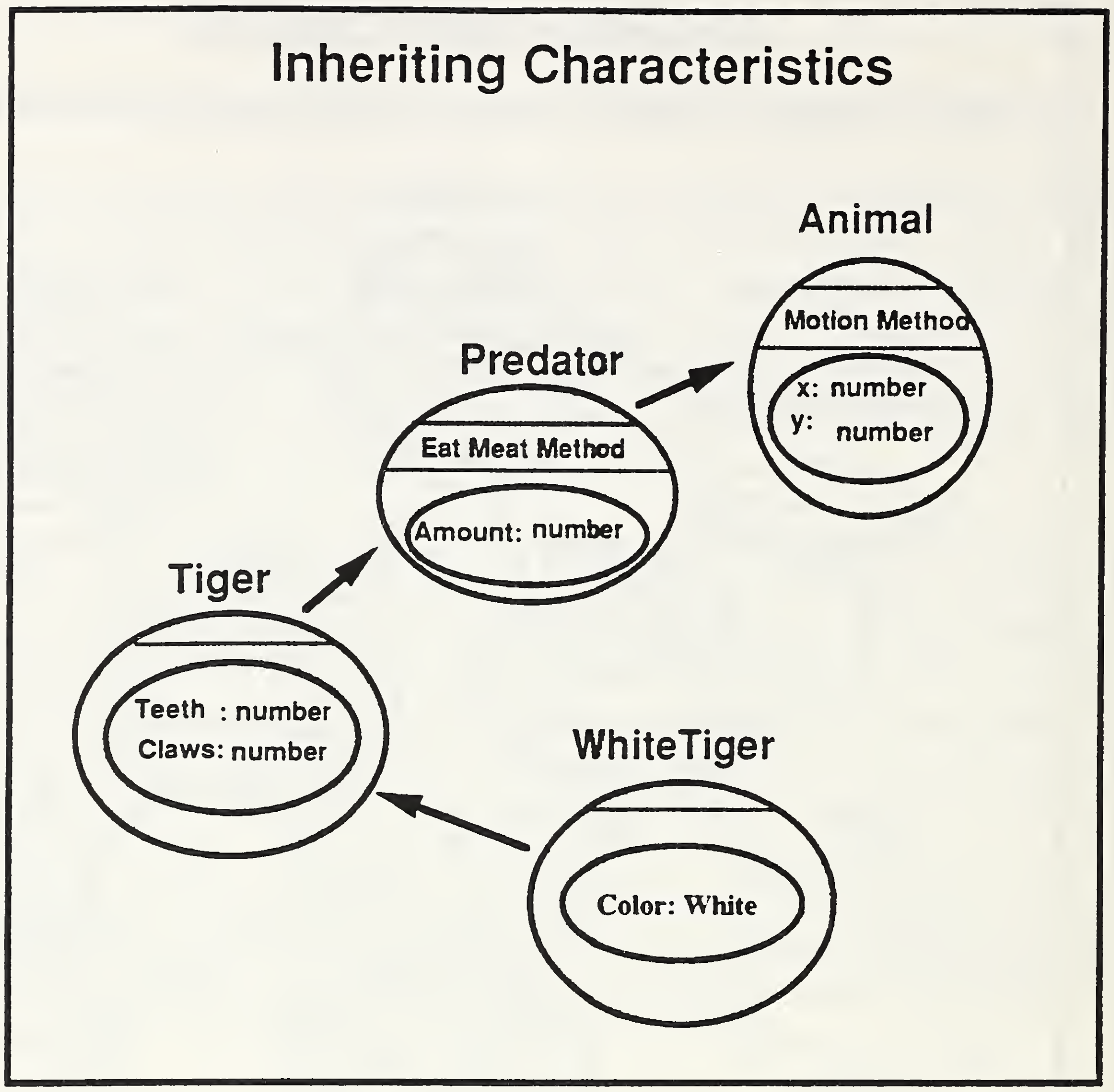


Figure 7. Automatic Inheritance Of Data and Methods

variables) of its parent classes. This facility greatly simplifies the definition and maintenance of complex hierarchies of classes and subclasses with all housekeeping performed transparently by the programming system as, for example, in the case of Smalltalk-80.

The object-oriented programming techniques discussed thus far serve primarily to off-load the programmer from the need to constantly think in terms of the details of the problem; it allows software development to be performed using the model's high-level objects, independent of their implementation details, and the messages they respond to. All processing, or object behavior, occurs when an instance of an object receives a message (roughly analagous to a subroutine call) and takes some action. This action

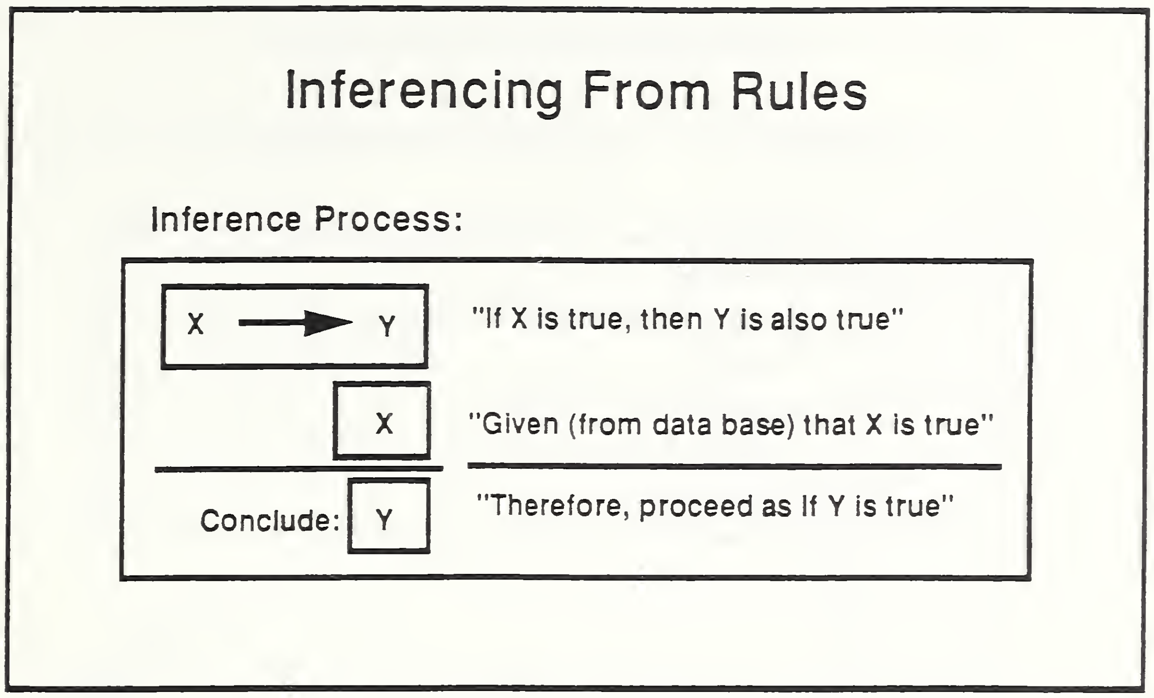

Inferencing From Rules
Inference Process:
X → Y
"If X is true, then Y is also true"
X
"Given (from data base) that X is true"
Conclude: Y
"Therefore, proceed as if Y is true"

Figure 8. Typical Inference Engine Process

can be the requested processing, or it can also result in other message sends to other objects in the system.

Rule-Based Inference. Rule-based inference extends the notion of high-level implementation one step further. It introduces ways to extend object behavior beyond the usual deterministic procedural computational methods programmed as fixed algorithms. Rule-based inference extends into each object's behavior open-ended noncomputational methods involving searching for a solution in a problem space. The advantage of this approach is that knowledge of a given problem domain can be expressed in very high level terms using "if--then" rules. This turns out to be quite similar to the ways people summarize rules of thumb about how to proceed to solve specific "common sense" problems without resorting to the details involved. Figure 8 illustrates the most common inference technique used in expert systems known as *modus ponens* from symbolic logic. The technique is deceptively simple, but powerful when combined with a large knowledge base of rules of the form shown in figure 9.

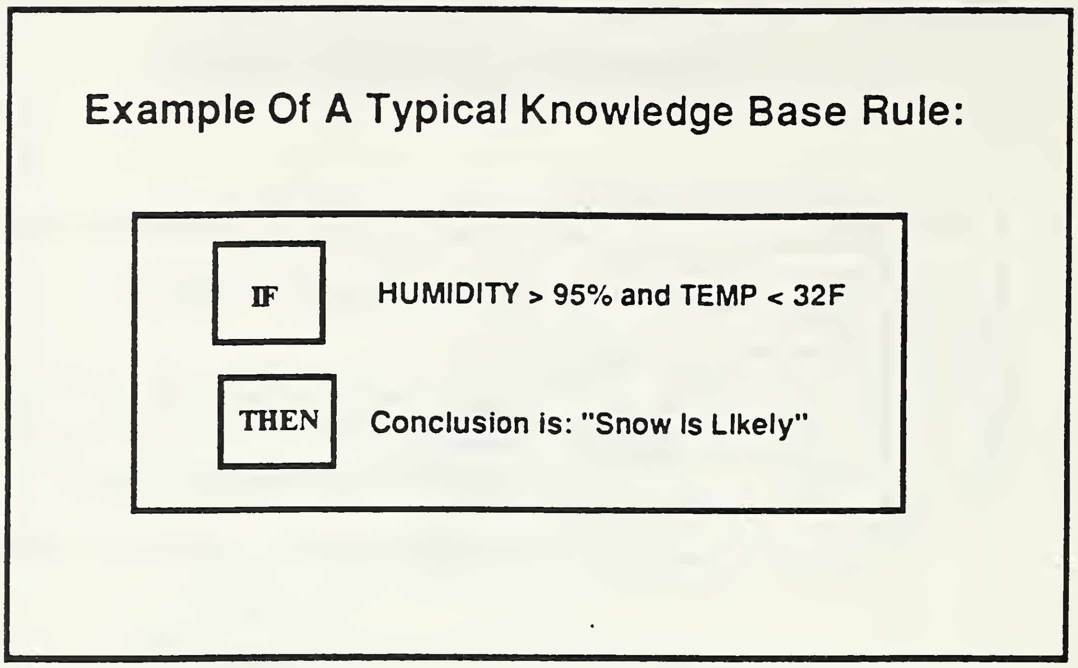

Example Of A Typical Knowledge Base Rule:
IF
HUMIDITY > 95% and TEMP < 32F
THEN
Conclusion is: "Snow Is Likely"

Figure 9. Capturing High-Level Knowledge As Rules

Such simple "If--Then" representations are adequate to capture the surface structure knowledge of a problem domain. Combined with other high-level representations, such as heuristics shown in figure 10, it is possible for a generalized program called an *inference engine* to chain through large sets of such rules as shown in figure 11, and reach conclusions not explicitly stored in the system, i.e., data base and knowledge base. The inference engine uses domain specific heuristics to reduce the scope of search, i.e., the number of rules that must be processed, to select the most

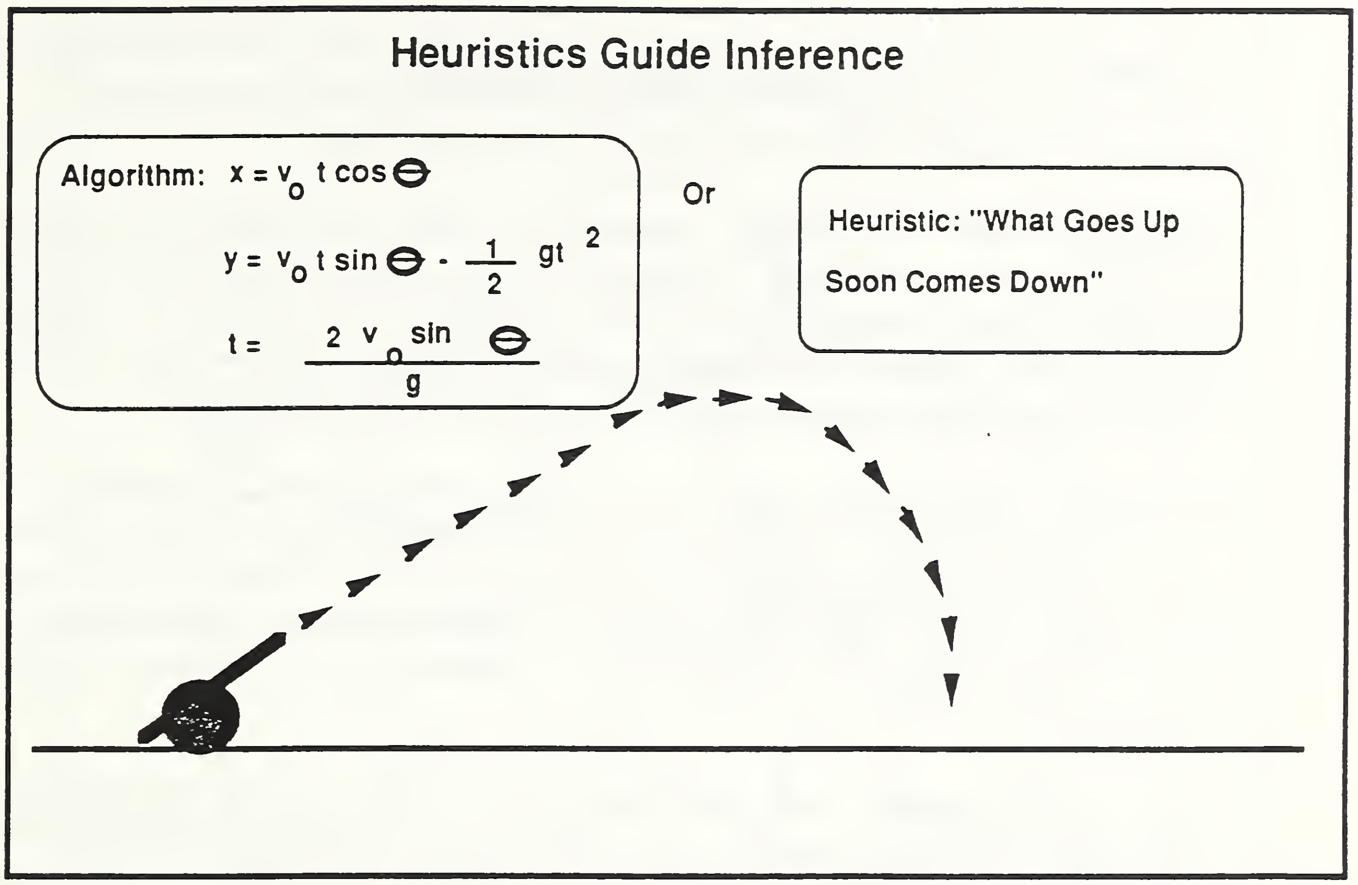

Heuristics Guide Inference
Algorithm: $x = v_o t \cos\theta$
$y = v_o t \sin\theta - \frac{1}{2} g t^2$
$t = \frac{2 v_o \sin\theta}{g}$
Or
Heuristic: "What Goes Up Soon Comes Down"

Figure 10. High-Level Heuristics Capture Knowledge.

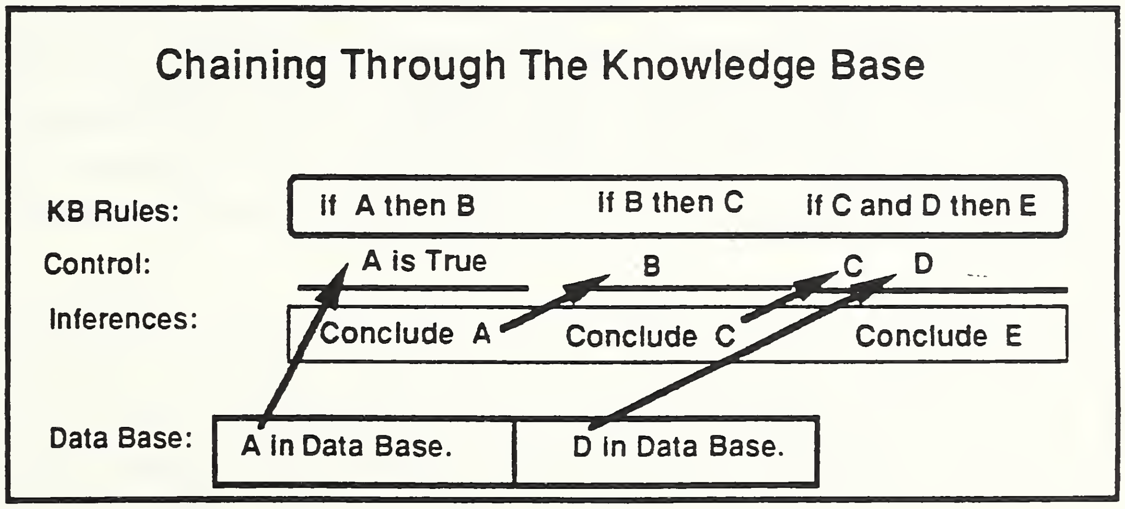


Figure 11. Program Flow Determined Through Inferences.

appropriate kind of search under the circumstances, or to increase the speed of the symbolic processing required to complete the reasoning chain and reach the final conclusions in minimum time.

Model-Based Reasoning. "If--Then" rules are extensively used in simple expert systems and have proven very effective in several problem domains where exclusive use of simple *surface* knowledge is adequate. However, the shallow knowledge captured in rules provides only a limited reasoning capability adequate to solve just the easiest problems.

For more complex tasks, those customarily thought of as requiring some *understanding* of the nature of the problem domain, the encoding of some amount of *deep* knowledge is necessary. As it turns out, object-oriented techniques are also quite appropriate for deep knowledge representation, as well as providing the high-level programming advantages already discussed.

Thus, an appropriate class and subclass hierarchy can be thought of as implicitly including a limited amount of deep knowledge structure of the problem domain being modeled, as illustrated in figure 12. This meta-knowledge about the class hierarchy itself, when coupled with the rule-based inference techniques discussed above and extended with recursive polymorphic messages to be introduced later, produces an very powerful programming methodology. This approach is appropriate for attacking complex problems having many more possibilities to consider than can be programmed in advance into a fixed algorithm using conventional techniques, such as those in nautical chart compilation. Although highly simplified for this discussion, the structure of the problem is similar to a family of communicating expert systems implemented as a set of cooperating objects, each covering the details of its problem area domain and each accessing sets of rules about its domain in the knowledge base.

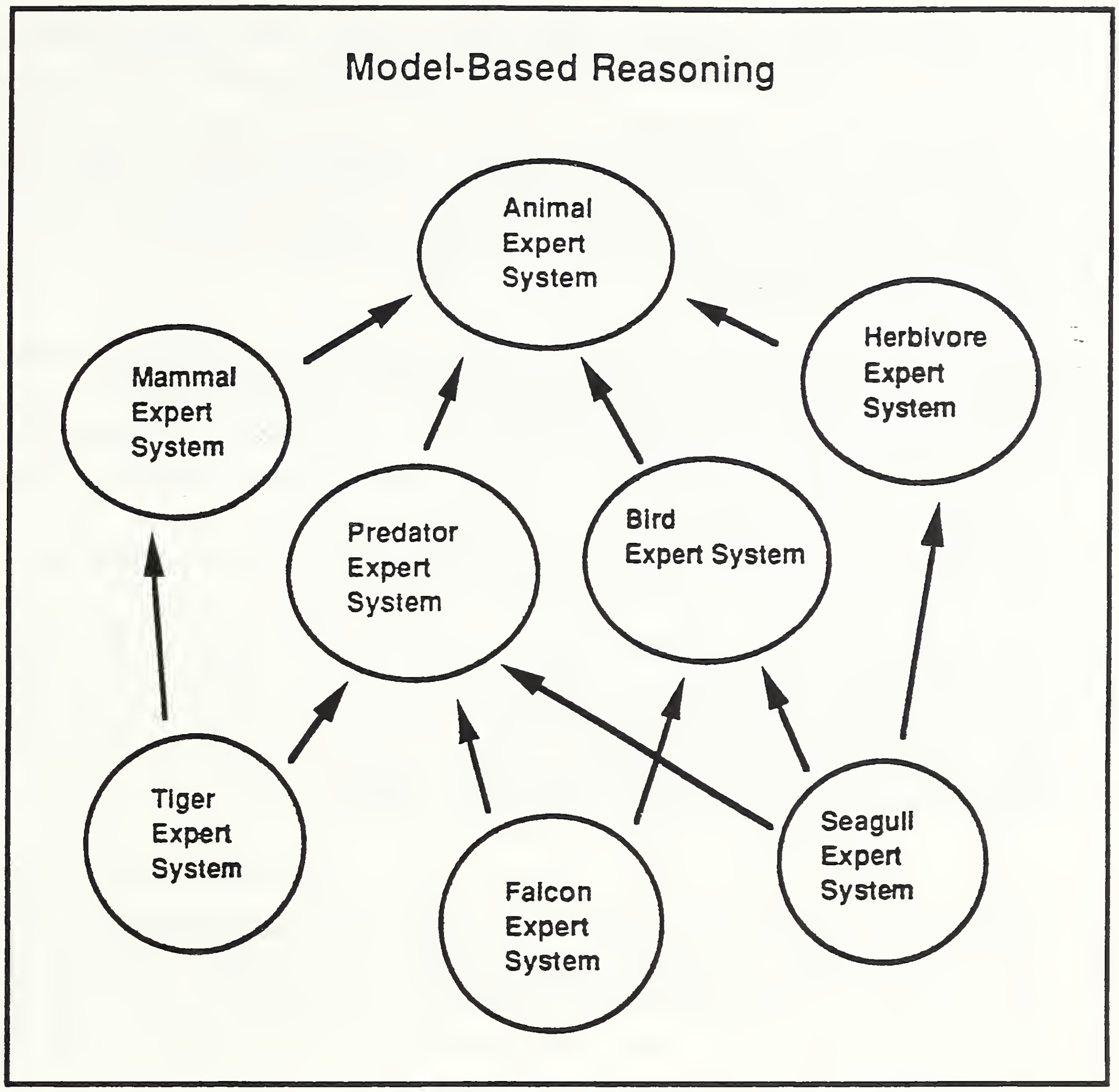


Figure 12. Model Establishes Context Information.

The heart of this new ability to attack problems of high complexity is the dynamic flow of program control, as already shown in figure 11, using model-based reasoning. The flow of processing, usually explicitly controlled by the programmer's design under conventional techniques, is determined dynamically during exection by the particular sequence of inferences reached. The next rule to be examined is determined almost entirely by the results of the previous inference and the constraints placed on the solution by the model.

Each path through the search space represents a set of possibilities to be examined and is different for every unique situation. The total of a problem solution is represented by the path of all possibilities through the search space. The search space has so many possibilities, in a difficult problem, as to make it virtually impossible to pre-program each one in advance, as would be required using fixed algorithms and conventional programming techniques.

Furthermore, the use of object-oriented techniques makes it possible to embed the symbolic reasoning capability of expert systems into some or all parts of the model, giving the effects of multiple expert systems, each handling a specific part of the model and returning its results to the others, as appropriate. NOAA's Chart Compilation Model, to be discussed in detail in another paper in these proceedings, operates in this manner .

Recursive Polymorphism. This fancy term denotes a closely related concept to model-based reasoning which is a marriage of the two techniques: (1) recursive procedures, or routines that repeatedly activate themselves, and (2) polymorphism, or multiple responses by objects in different classes (data types) to the same message.

NOAA's model-based processing for chart compilation depends upon the ability of the system to automatically send identical messages to different parts of the model and have each respond in a manner appropriate to its class of objects. Moreover, the processing initiated is recursive in that a composite object, i.e., one composed of nested lower-level objects, automatically performs the processing operation on itself and any and all lower-level objects that it may contain. This type of processing is the key to designing and implementing such complex software systems as GIS.

For example, a shipping channel may be requested to position itself on a chart according to the rules in the knowledge base about channels. In order to accomplish this task, the channel relays the same message to all of its channel segments who, in turn, relay the message to their buoys, lights, soundings, hazzards, and accompanying text. These lowest-level entities perform their individual processing, as appropriate to their context, and return the results up the recursive chain of control until all processing is completed and the channel is properly positioned on the chart.

The Embedded Expert System Concept. The traditional view of an expert system is one in which a user consults the system through a terminal or personal computer for advice. Classical expert systems are designed to ask the user to respond to a sequence of questions. Then the system chains through its knowledge base to infer one or more conclusions about the problem domain, given the rule parameter values entered.

This form of system provides expert level advice to a person who then uses the provided information to carry out some remaining sequence of actions manually. In this case, the user is quite aware that an expert system is being consulted and that subsequent actions to be taken are based upon the advice from such a system. It is likely that complete use of the system requires specialized training on selected details of how the inferencing is done, or the structure and content of the rules in the knowledge base. The more advanced systems may even query an internal or remote data base for parameter values, rather than request that all values be entered by the user.

An embedded expert system carries this idea a step farther. Instead of requiring the user to provide detailed parameter values and subsequently carry out further steps to complete an activity, the expert system returns its results to a higher level conventional program, for example, which completes the activity in a conventional manner. This concept is illustrated in figure 13. The person is using a workstation which, for particular situations, consults one or more embedded expert system modules and then uses the answers it receives to complete normal processing.

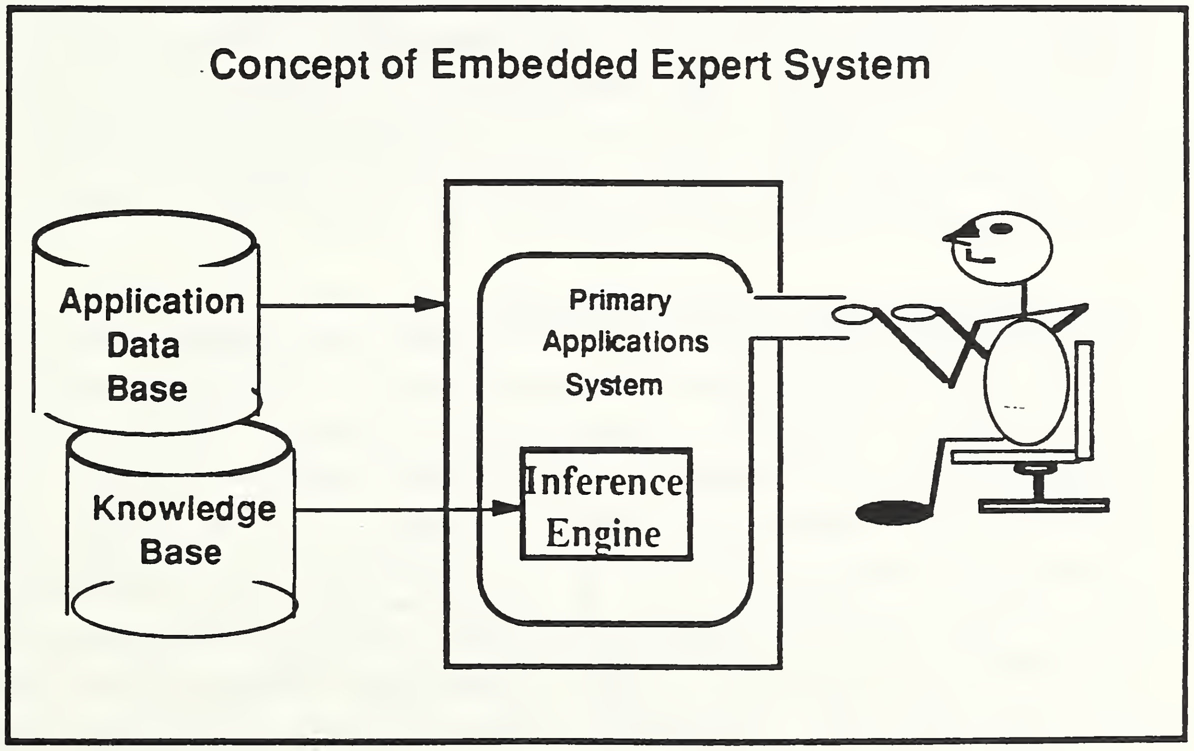

Concept of Embedded Expert System
Application Data Base
Knowledge Base
Primary Applications System
Inference Engine

Figure 13. Actions Of Expert System Transparent To User

In this case, the user does not have to be aware that an expert system is involved; the only skills needed are how to initiate the workstation's activities and how to use the facilities provided to get the work done. This implies that the user in not required to be specially trained or skilled in using expert systems, knowledge bases, inference engines, etc., *per se*; the user only has to be able to operate the workstation and use its functions as tools to carry out expected work tasks. These are the kinds of expert systems we are speaking of when we refer to enhancing a primary system with embedded expert system tools.

The CartoAssociate Prototype.

The CartoAssociate prototype is an attempt to combine the results of the prototypes listed in Table 1 under a coherent and integrated design. Its components include: (1) the hierarchial object model system design; (2) the general chart compilation model (CCM); (3) the spatial object data model; (4) the knowledge base; and (5) the feature data base. Although space does not permit an in-depth technical discussion, an overview of these key ideas and their enabling technologies can be briefly described.

The Hierarchial Object Model System Design. The fundamental principle for the overall design is maximum exploitation of the object-oriented paradigm using object class hierarchies enhanced with intelligent behaviors at all levels. This means that each component is an object with its own set of intelligent behaviors as supplied by an expert system object and knowledge base object. The collective behavior of all objects represents the total processing required to compile NOAA nautical charts.

<u>The fundamental design principle:</u> As depicted in figure 14, chart compilation processing is initiated when the cartographer sends a message to a specific NOAA chart to compile *itself*. The chart is an intelligent object with the behavior of compiling its representation from the current version of the chart feature data base when requested to do so. That is, it *knows* how to detect and resolve feature conflicts, change scale and generalize features, choose correct symbols, and correctly place text according to chart-level rules in the knowledge base. The chart object accomplishes this by sending polymorphic messages recursively down the data structure hierarchy to those subordinate features in the object data base affected by the compilation. They are instructed to *place themselves* properly without conflicts, etc., according to their rules in the knowledge base. They, in turn, relay corresponding messages to their subordinate

features. This recursive message sending continues until the bottom of the hierarchy is reached and processing is completed.

Some features will be composite features, that is, features made up of other composite and simple features, and so on, until the lowest level primitive features process themselves. As previously discussed, for example, a ship channel is a typical composite feature. The channels are composed of channel segments, each of which is further composed of defining features such as buoys, lights, signals, and depth curves. Channel segments also contain other simple features within its geographical space, such as critical soundings and individual wrecks, rocks, pilings, pipes, cables, and other hazards.

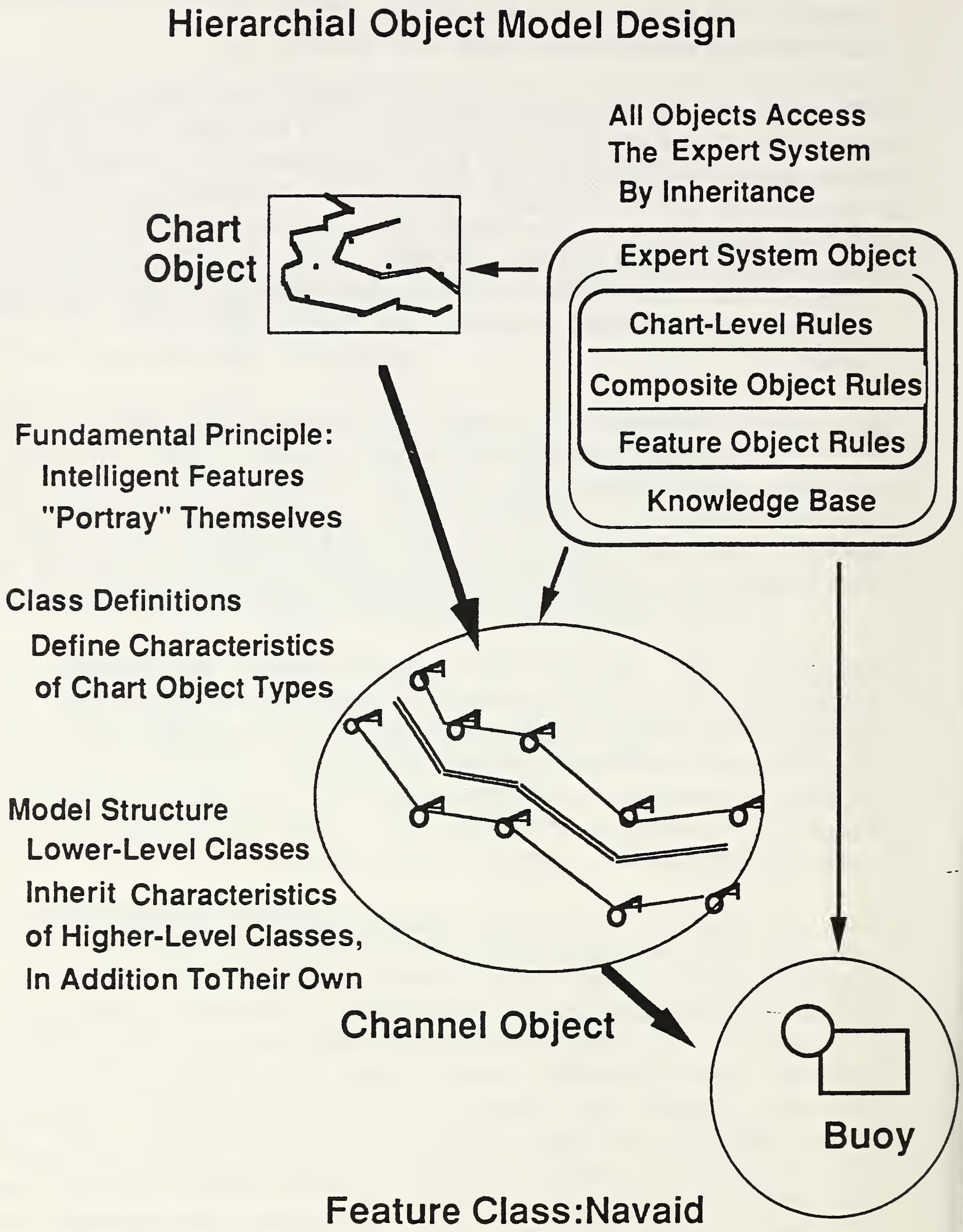

Hierarchial Object Model Design
All Objects Access The Expert System By Inheritance
Chart Object
Expert System Object
Chart-Level Rules
Composite Object Rules
Feature Object Rules
Knowledge Base
Fundamental Principle: Intelligent Features "Portray" Themselves
Class Definitions Define Characteristics of Chart Object Types
Model Structure Lower-Level Classes Inherit Characteristics of Higher-Level Classes, In Addition ToTheir Own
Channel Object
Buoy
Feature Class:Navaid

Figure 14. Processing By Recursive Polymorphic Messages

The net effect of this model-based system recursively propagating messages down its hierarchy results in each chart, composite feature, and simple feature behaving as a family of coupled embedded expert systems, each contributing to the solution of the overall problem at its own level of detail to produce the final result, i.e., the correct compilation of a particular NOAA navigational chart.

The General Chart Compilation Model

Figure 15 is a view of the General Chart Compilation Model's structure. It illustrates the recursive descent message passing for *evaluation processing* which is the first step in the model's operation. Evaluation processing is the pass in which features of the same type attempt to depict themselves without regard to their proximity to other types of features. The *specification* contains information on the details of the particular chart's data contents: scale, geographic coverage, and other chart specific details. The *skeleton* contains nonfeature information appearing on the chart, such as compass roses and textual annotations. The *data base root node* is the entry point into the Spatial Object Data Model's feature object compilation data structure. This structure contains features in object form to be depicted on the chart, or ones that may have an effect on the compilation processing in some other as yet unanticipated (to the evaluation processing step) manner. The *virtual chart root node* is the entry point into the data structure which will hold the results of step 1 evaluation processing.

The results of step 1 is a tree data structure called a preliminary *Virtual Chart* whose leaves are *Symbology Objects* corresponding to the chart's features as found in the spatial object data base. The correspondence is not necessarily one-to-one, however, as some feature objects may be combined into a single symbology object, as in the case of feature coalescence upon changing scale, e.g., a group of buildings may be combined into a single symbol when moving to a smaller scale (larger area). Symbology objects also have access to the inference engine and knowledge base and, therefore, implement intelligent behaviors required for the next processing step.

Figure 16 continues the simplified view of the structure of the general Chart Compilation Model. It illustrates the recursive ascent (i.e., the unwinding of the recursion) message passing for *application processing*, which is the second step in the model's operation. In this pass, the symbology objects generated in step 1, whose tree data structure is pointed to by the virtual chart root node, attempt to finalize their depiction while considering their proximity to other nearby features of different types which may possibly be in conflict, or whose presence may influence the choice of final symbology at the scale being used.

Upon completion of step 2's application processing, the result is a completed virtual chart with scale changed, features generalized, conflicts resolved, symbols selected, and text placed. This virtual chart, however, is not simply a graphic file, but an intelligent object with its own set of behaviors. It is a tree data structure populated by finalized symbology objects which becomes "real" when sent a message by the cartographer to present itself as a screen display or final plot. Only at that time does the virtual chart transform itself from an abstract data structure to assume a concrete, physical form.

More detailed discussion of the compilation model is too involved for the purpose of this paper. Suffice it to say that the GCM can be decomposed into the following parts: (1) the GIS scale change/generalization model (GSC/G) (refer to the paper by Thorn and Morgan in this proceedings); (2) the conflict detection/resolution (CD/R) model; (3) the text placement/labeling model (TP/L); and (4) the decluttering/sounding selection model (D/SS). The CD/R and TP/L are being developed in-house and the GSC/G and D/SS are being developed under contract.

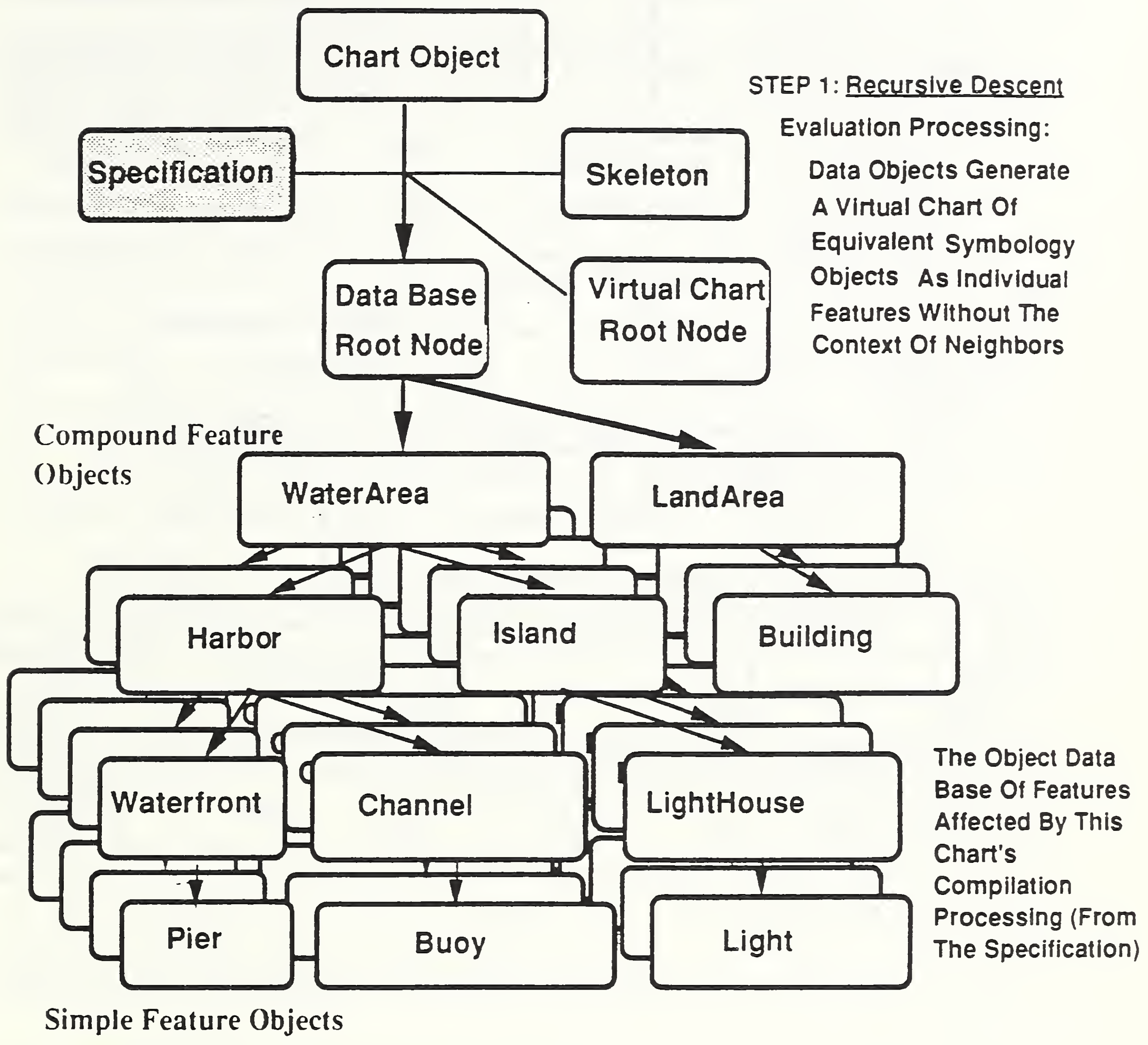


Chart Object
Specification
Skeleton
Data Base Root Node
Virtual Chart Root Node
STEP 1: Recursive Descent Evaluation Processing: Data Objects Generate A Virtual Chart Of Equivalent Symbology Objects As Individual Features Without The Context Of Neighbors
Compound Feature Objects
WaterArea
LandArea
Harbor
Island
Building
Waterfront
Channel
LightHouse
Pier
Buoy
Light
The Object Data Base Of Features Affected By This Chart's Compilation Processing (From The Specification)
Simple Feature Objects

Figure 15. The Compilation Model's Evaluation Processing

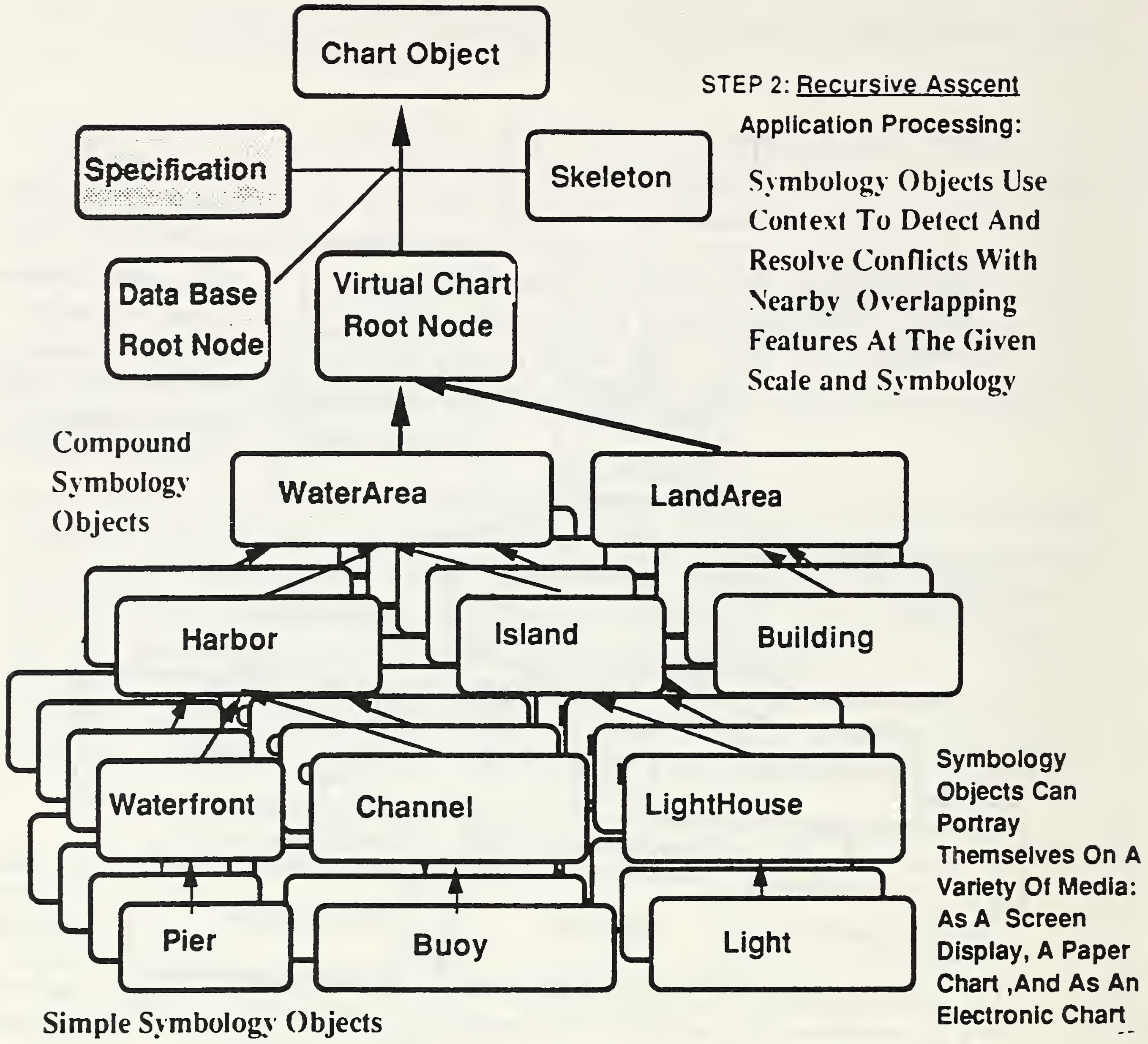


Chart Object
STEP 2: Recursive Asscent
Application Processing:
Symbology Objects Use Context To Detect And Resolve Conflicts With Nearby Overlapping Features At The Given Scale and Symbology
Specification
Skeleton
Data Base Root Node
Virtual Chart Root Node
Compound Symbology Objects
WaterArea
LandArea
Harbor
Island
Building
Waterfront
Channel
LightHouse
Symbology Objects Can Portray Themselves On A Variety Of Media: As A Screen Display, A Paper Chart ,And As An Electronic Chart
Pier
Buoy
Light
Simple Symbology Objects

Figure 16. The Compilation Model's Application Processing

The Spatial Object Data Model.

Figure 17 illustrates the concept of the Spatial Object Data Model which was developed to support the Chart Compilation Model's detailed processing just described. This new data model combines the virtues of traditional cartographic-spatial models with those of the emerging object-entity data models. This new model was developed to overcome the serious limitations of the earlier models used in GIS and automated cartography.

The most serious of these limitations is the inability of existing data models to capture the "is composed of" relationship of compound features, as outlined above by the shipping channels on NOAA's nautical charts. The popular relational models depend upon data normalization, or the elimination of redundant data fields, in the various tables (relations) used to store specific kinds of data. This normalization operation, while necessary for the relational model's referential integrity and highly efficient for simple data files, effectively removes all the "intelligent" relationships required for processing compound spatial feature data.

The subsequent retrieval of relational data requires that these semantic relationships first be restored. This is accomplished by multiple "joins" of all required relations using foreign keys (a single field common to both tables) as links. This, in turn, requires that long sequences of SQL (Structured Query Language) commands be written by the system user, or generated by very complex internal software, in order to define the specific joins required. This additionally assumes that all the joins to be needed for future processing can be anticipated in advance and coded into the system. In either case, the retrieval operation, which may require tens or even hundreds of joins of separate tables to restore a single highly complex nested feature, exacts a very large processing overhead and, correspondingly, imposes long response penalites on the system.

By contrast, the object data model organizes and stores data entities directly by name and retains their internal semantic relationships as described to them by the designer, with pointers into the various subunits that make up

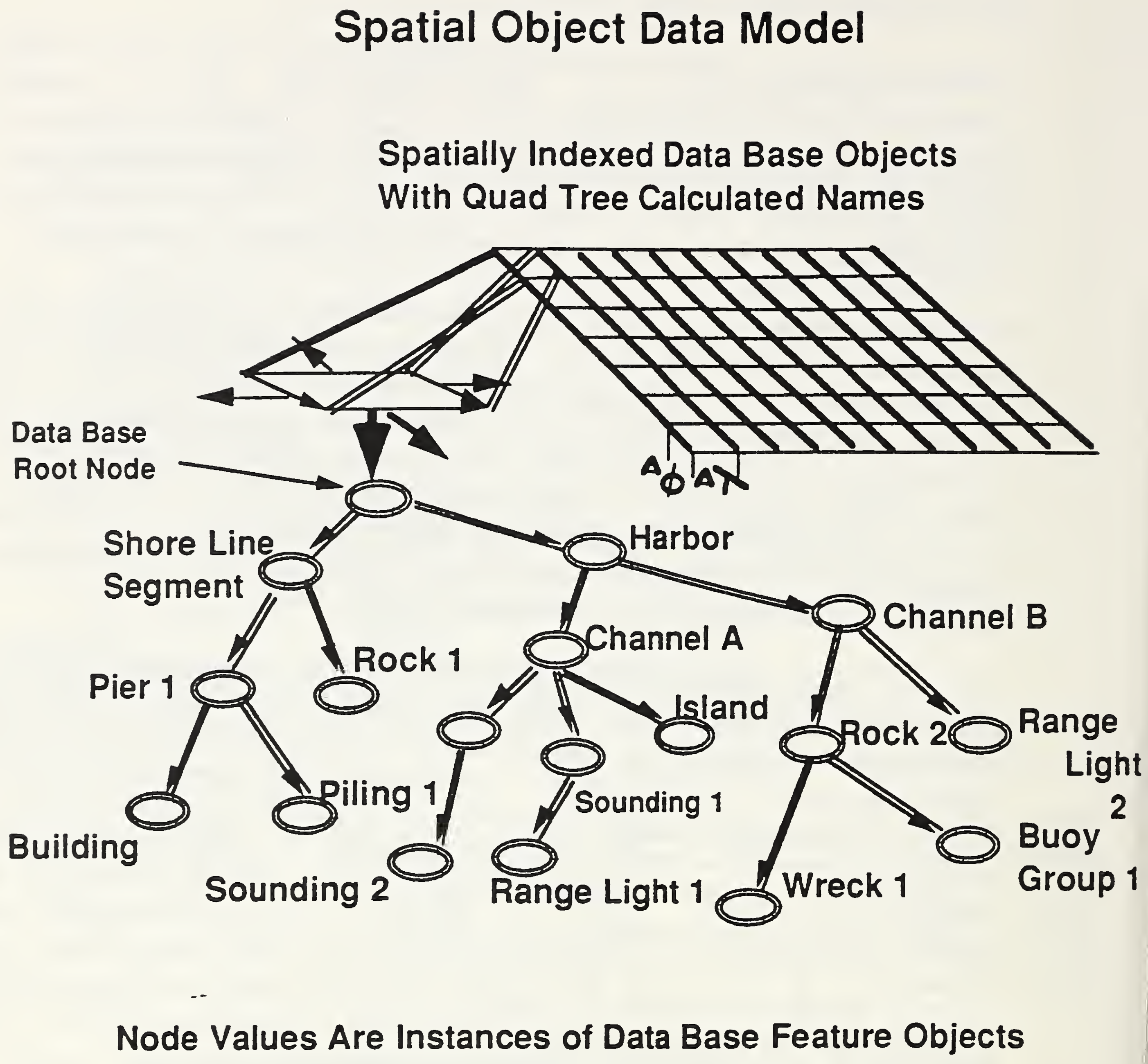


Figure 17. Example Object Data Model Tile Tree

any compound entities. Thus, the object data base system can quickly identify and reference a data object and all its subobjects without the need to perform lengthy intermediate restorative operations for internal semantic relationships. This spatial version of the object data model goes even further to provide a way of representing the complex spatial relationships required of GIS systems.

This unique data model improves upon a spatial data base's ability to retain and represent, in a software sense, both the complex spatial relationships and the compound feature relationships required for this special type of data processing. To our knowledge, this is the first time such object data base techniques have been considered and applied in this manner to GIS spatial data base support.

The Spatial Object Data Model has two primary components: (1) a set of *tiles* defined as data base objects, and (2) a *tree data structure* for each tile populated with the data base feature objects corresponding to the available chart feature data for the geographic area of the tile. The complete set of tiles constitutes the feature data for the geographic area of the data base.

Each tile's object name (entities in object data bases are stored and referenced only by their object names) is generated by a variable resolution quad-tree indexing scheme based upon the data density and the latitude-longitude area the tile covers. Thus, the individual tile entities are related to one another and assume spatial significance due to the method used to calcualte their data base identifiers, i.e., their names. Every tile in the set is linked to its four neighbors by pointers, thereby enabling spatial references-into the data base for any data base query to quickly reach all tiles covering the geographic area of any given feature.

The tree structure for each tile is accessed through that tile's data base root node. By following the pointers through the tree, the system can reach the nodes, which are individual instances of data base feature objects. Significantly, even in the data base, the chart features have the same set of intelligent object behaviors as they do when used at the workstation. Thus, it is possible to trigger a significant amount of processing *in the data base itself*, not limiting the system to the usual simple data base queries for data retrieval. Thus, *results of intelligent processing*, rather than large volume data retrievals, are transmitted over the network to the workstations, thereby minimizing network traffic and lowering bandwidth requirements.

The Knowledge Base and Intelligent Objects

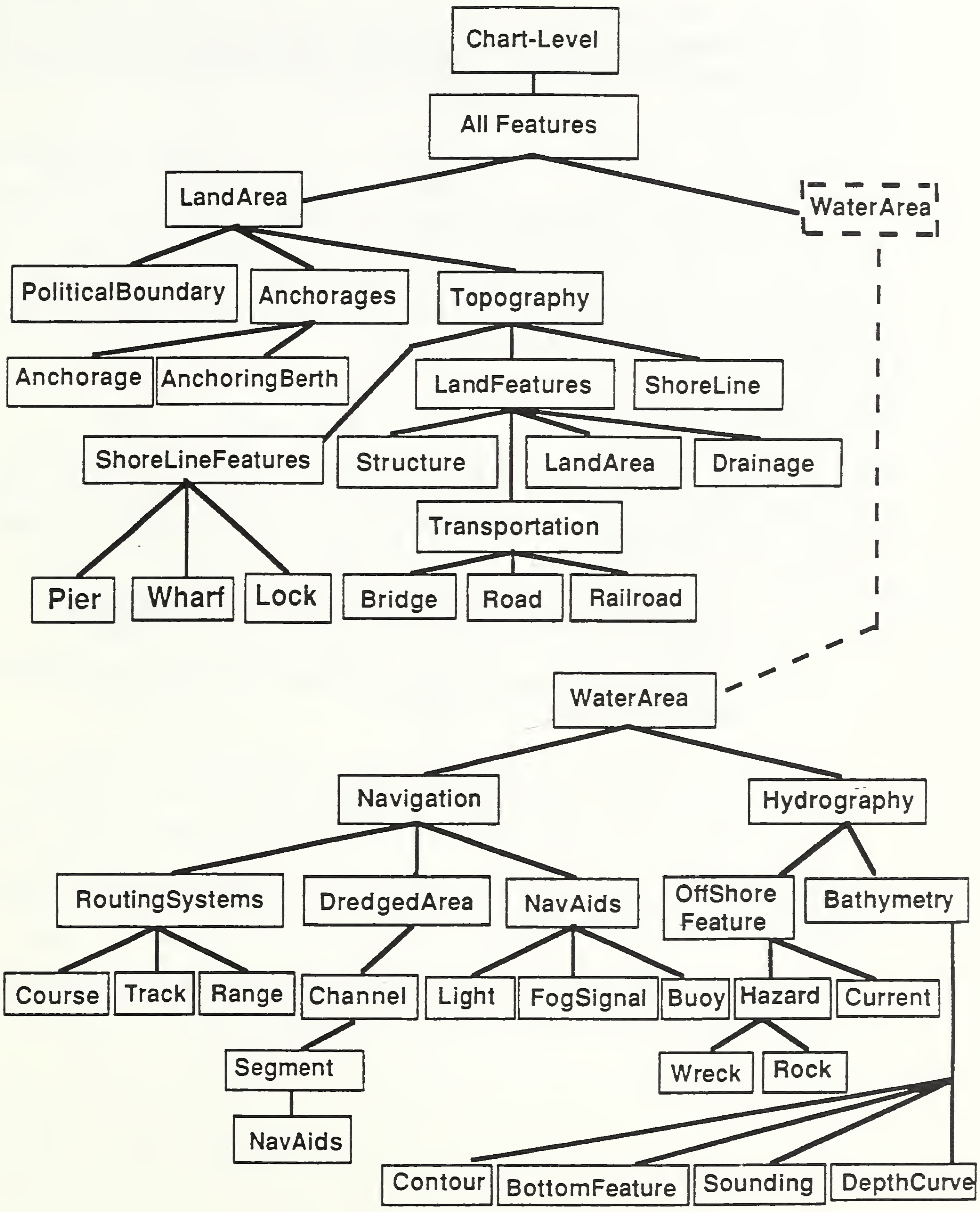
The Knowledge Base is the collection of "if--then"style rules applied by the feature and symbology objects by way of the expert systems's inference engine to effect intellegent processing. Rather than one very large set of rules, expert systems usually organize rules into a hierarchy of subsets called "frames" which are in some correspondence to the structure of the problems being solved.

For example, if the knowledge base is one about diagnosing the failure of, say, an automobile engine, there will be a top level engine frame (set of rules for overall engine characteristics). Below that, there will be frames for the electrical system, the fuel system, the steering system, and the transmission and drive system. In this manner, not all knowledge base rules will be searched each time the inference engine is activated. Only the rules in the applicable frames will be searched, making for more efficient and timely processing.

The CartoAssociate's knowledge base is decomposed into frame-like object classes which parallel the structure hierarchy of the families of features found on NOAA's nautical charts. These rule classes, depicted in simplified form in figure 18, like all objects in the system, can inherit more general rules from their super classes, thereby effecting considerable compaction of the knowledge base, since rules that apply to more than one feature are not duplicated.

The more general rules are found near the top of the hierarchy and highly specific rules are found at the bottom of the hierarchy. In this manner, any feature or symbology object has direct access to general or highly specific rules that are appropriate for resolving evalutaion-compilation problems about chart features in its class (e.g., two ship wrecks in conflict) or for

Figure 18. Nautical Feature/Knowledge Base Class Hierarchy



application-compilation problems about chart features between classes (e.g., a ship wreck, a buoy, and a sounding in conflict). The more general rule classes at the top of the hierarchy apply to chart-level processing, and serve as a check that the results produced by lower-level processing remain appropriate at the overall chart level.

Notice in figure 18 that "NavAids" appears twice, once under "Navigation" and again under "Channel -- Segment." This is a form of inheritance known as "multiple inheritance" in which a subclass inherits characteristics from two or more parent classes. In this case, navigational aids which are part of a compound object "channel - segment" also inherit all the general characteristics of isolated navaids as well as characteristics unique to those which define a channel segment's boundary.

Intelligent Objects

To complete this overview of the CartoAssociate, figure 19 depicts the technique used to implement intelligent behavior among the systems's chart feature and symbology objects which was alluded to in figure 14. Chart features, being in object form, are of two parts:

(1) encapsulated data - the values of the instance variables, and

(2) methods - the program code available to the object to manipulate its data fields (instance variables).

The data is considered to "encapsulated" because it is isolated from manipulations by other system objects; only the object's own methods can access it. The methods are Smalltalk-80 program statements which cause all of the object's processing to occur. They consist of references to instance variables and even send messages to other objects to obtain values used in subsequent processing.

Consequently, when the flow of control reaches a Smalltalk-80 statement which queries the expert system object, a message send to the inference

Figure 19. Adding Intelligence To Objects

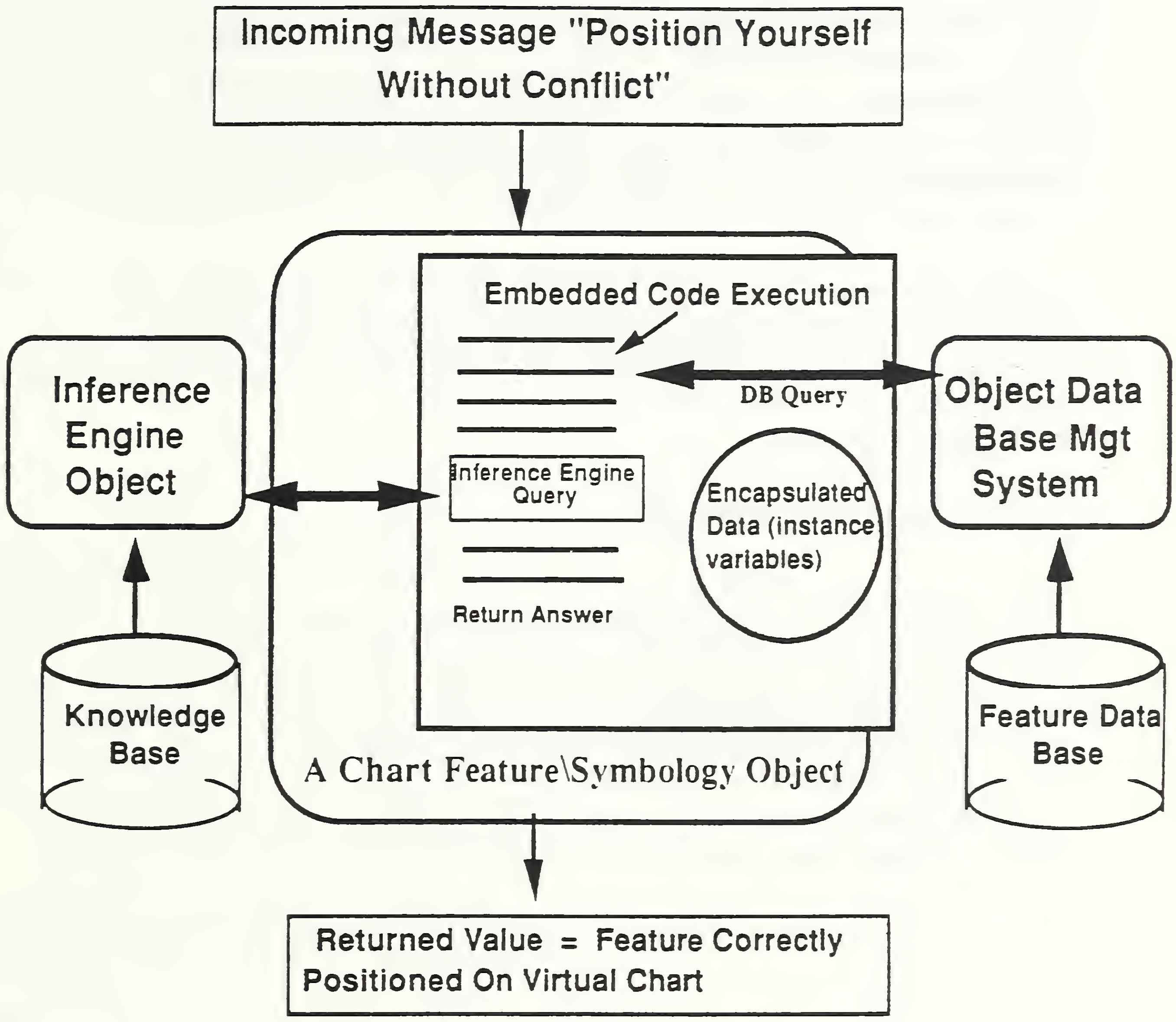

Incoming Message "Position Yourself Without Conflict"
Embedded Code Execution
Inference Engine Object
DB Query
Object Data Base Mgt System
Inference Engine Query
Encapsulated Data (instance variables)
Return Answer
Knowledge Base
Feature Data Base
A Chart Feature\Symbology Object
Returned Value = Feature Correctly Positioned On Virtual Chart

Chart Features Become A Family Of "mini" Expert Systems Working In Concert To Achieve Intelligent Behaviors For Conflict Detection/Resolution, etc.

engine object occurs and the method waits for the value returned from the send. No distinction is made between a message send to the inference engine and one to any other object in the system. The intelligent behavior occurs because the value returned from the inference engine call is one which has been deduced via symbolic computation, i.e., chaining through the knowledge base. When this result is returned to the method, it resumes normal flow of control until the routine exits and returns to its caller.

In this manner, every feature in the data base has access to the expert system and becomes, in effect, a miniture specialized expert system for its own set of functions. In addition, this family of expert systems works in cooperation to achieve the total set of effects required to solve the chart compilation problem.

Summary and Conclusions

This tutorial paper has presented an overview of NOAA's expert system prototype for compiling nautical navigational charts, the CartoAssociate. An introductory discussion summarized chart compilation problem fundamentals as well as object-oriented and expert systems concepts used in the prototype's implementation. The prototype's overall design was discussed in terms of its intelligent object hierarchy, compilation and spatial object data models, knowledge base/feature data base structure, and the techniques used to add intelligence to NOAA's chart feature objects.

The results to date indicate that the design and implementation strategies work, as evidenced by the live demonstration provided at this Workshop. Much remains to be done, including volume testing of the object data base and expansion of the knowledge base. The question remains as to whether the prototype will scale-up performance-wise when large volumes of data and the full complement of chart feature types are included. Shifting more processing into the GemStone object data base may also be another option.

We feel that the validity of the approach of integrating GIS techniques and an expert system with an object-oriented platform (programming environment, language, and data base) to solve heretofore intractable chart compilation problems has been proven. In a wider context, this work demonstrates that the path toward finished graphic depictions from spatial GIS queries lies in the object-oriented language/data base and expert systems arena.

References

1. Bossler, John D., et al., 1988. *Knowledge-Based Cartography: The NOS Experience*. The American Cartographer, Vol.15, No.2.149-161.

2. Drinnan, Charles H., Mattair, Charles G., Jr, and Luckey, Stephen E., 1989. *An Interactive Expert Editing System For Nomenclature Placement*. Proceedings of the ASPRS/ACSM, Baltimore, Maryland, 5,p.221.

3. Lisle, James L., 1988. *Automated Nautical Charting System II*. Proceedings of the U.S. International Hydrographic Conference, Special Publication No. 21, April 12-15, 1988, Baltimore, Maryland, 167-170.

4. Luckey, Stephen E., Pendleton, Dave., and Walton, Fred, 1987. *Knowledge-Based Cartography: Capturing Cartographic Expertise*. Proceedings of the ASPRS/ACSM Annual Convention, March 29-April 3,1987, Baltimore, Maryland,3,154-161.

5. Luckey, Stephen E., and Pendleton, Dave., 1988. *Object-Oriented Concepts As Applied To Automated Chart Compilation*. Proceedings of the U.S. International Hydrographic Conference, Special Publication No. 21, April 12-15, 1988, Baltimore, Maryland, 171-177.

6. Skylar Technology, Inc., 1988. *The Feasibility of Implementing An Object-Oriented Expert System Shell As A Front-End To An Object-Oriented DBMS*. Final Report, National Bureau of Standards Contract Number 43NANB814788, Skylar Technology, Inc., Glen Burnie, Maryland, 7pp.

7. Skylar Technology, Inc., 1988. *CartoAssociate Expert System Design Concept*. Final Report, National Bureau of Standards Contract Number 43NANB814788, Skylar Technology, Inc., Glen Burnie, Maryland, 9pp.

8. Thorn, Matthew., et al., 1989. *Spatial Knowledge Base Management System For Mapping and Charting*. Final Report, NOAA/NOS Contract Nol 50-DKNA-9-00124, Spatial Data Sciences, Inc., McLean, Va., 40pp.

NIST-114A (REV. 3-90) U.S. DEPARTMENT OF COMMERCE NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY	1. PUBLICATION OR REPORT NUMBER NISTIR 4703
BIBLIOGRAPHIC DATA SHEET	2. PERFORMING ORGANIZATION REPORT NUMBER
	3. PUBLICATION DATE OCTOBER 1991

4. TITLE AND SUBTITLE

Technology Integration Workshop: Selected Papers

5. AUTHOR(S)

Henry Tom (editor)

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS) U.S. DEPARTMENT OF COMMERCE NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY GAITHERSBURG, MD 20899	7. CONTRACT/GRANT NUMBER
	8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

GIS Standards Laboratory
CSL
NIST
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

This report contains selected summaries of technical presentations and demonstrations given at the National Institute of Standards and Technology (NIST) Geographic Information Systems (GIS) Standards Laboratory's Technology Integration Workshop held at NIST, Gaithersburg, Maryland on August 23-24, 1990. The Workshop hosted over 50 representatives from a wide variety of governmental, industrial, and academic organizations and generated considerable interest and discussion among the attendees.

The Technology Integration Working Group, chaired by Mr. Dave Pendleton, NOAA, was formed within the GIS Lab as a cooperative technology transfer vehicle to share advances being made in applying expert systems, object-oriented database technologies, and GIS to practical problems in spatial data management and cartographic portrayal. The Workshop focused upon demonstrations of work-in-progress prototypes and technical discussions of progress in several on-going projects.

This Workshop as well as other NIST GIS Standards Laboratory activities are focused on performing cooperative research in order to integrate existing, emerging, and the anticipatory development of spatial data and information technology standards. The forum of government, industry, and academic organizations participating in the NIST GIS Standards Laboratory is dedicated to achieving this objective.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

technology integration; GIS; expert systems; object-oriented technology; geographic information systems; digital cartography

13. AVAILABILITY	14. NUMBER OF PRINTED PAGES
☐ UNLIMITED	
[✓ X] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).	
☐ ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.	15. PRICE
☐ ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.	

ELECTRONIC FORM